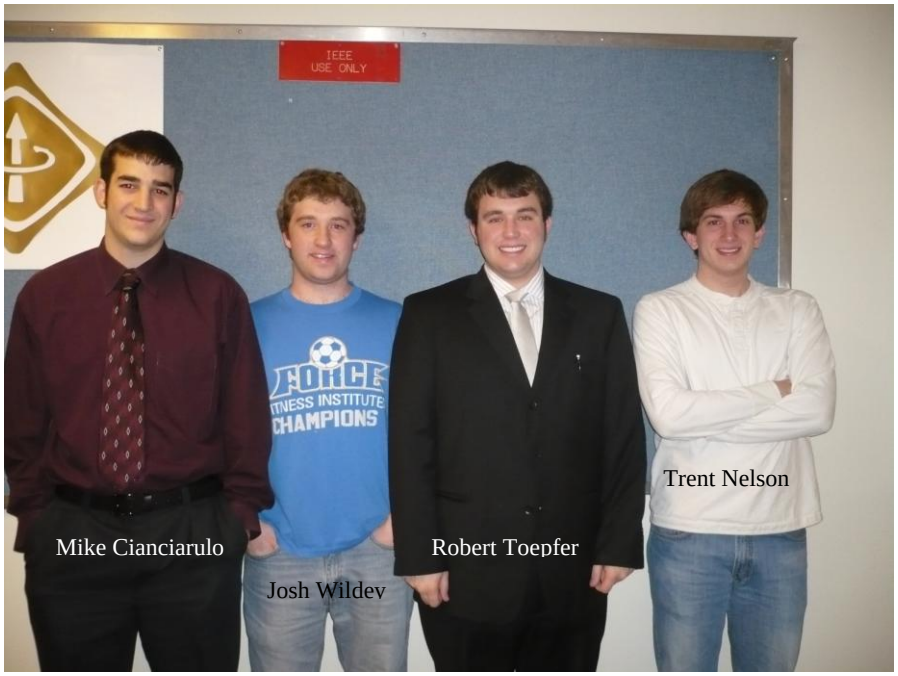
ECE 477 Final Report – Spring 2008
Team 8 – OMAR



Team Members:

#1: _____ Signature: _____ Date: _____

#2: _____ Signature: _____ Date: _____

#3: _____ Signature: _____ Date: _____

#4: _____ Signature: _____ Date: _____

CRITERION	SCORE	MPY	PTS
Technical content	0 1 2 3 4 5 6 7 8 9 10	3	
Design documentation	0 1 2 3 4 5 6 7 8 9 10	3	
Technical writing style	0 1 2 3 4 5 6 7 8 9 10	2	
Contributions	0 1 2 3 4 5 6 7 8 9 10	1	
Editing	0 1 2 3 4 5 6 7 8 9 10	1	
Comments:		**TOTAL**	

TABLE OF CONTENTS

Abstract 1
1.0 Project Overview and Block Diagram 2
2.0 Team Success Criteria and Fulfillment 3
3.0 Constraint Analysis and Component Selection 4
4.0 Patent Liability Analysis 11
5.0 Reliability and Safety Analysis 14
6.0 Ethical and Environmental Impact Analysis 18
7.0 Packaging Design Considerations 21
8.0 Schematic Design Considerations 25
9.0 PCB Layout Design Considerations 27
10.0 Software Design Considerations 29
11.0 Version 2 Changes 32
12.0 Summary and Conclusions 33
13.0 References 34
Appendix A: Individual Contributions A-1
Appendix B: Packaging B-1
Appendix C: Schematic C-1
Appendix D: PCB Layout Top and Bottom Copper D-1
Appendix E: Parts List Spreadsheet E-1
Appendix F: Software Listing F-1
Appendix G: FMECA Worksheet G-1

Abstract

OMAR is part of a solution to the Purdue IEEE Aerial Robotics' project which is an ongoing project and competes in the annual International Aerial Robotic Competition (IARC). Its primary function is to provide autonomous reconnaissance within an unexplored room. It is going to be a land based wheel driven vehicle which will use an array of range finders and proximity sensors to autonomously navigate and map out a room while avoiding obstacles. In addition to that it will also have a camera which will take still images. OMAR will continue to navigate this room until a specified logo is identified and relayed wirelessly back to a base station, in this case, a laptop computer. OMAR has been able to achieve autonomous navigation and logo detection but was unable to complete room mapping. Regardless of the results it is still fully capable of completing the intended mission.

1.0 Project Overview and Block Diagram

OMAR is the Outstanding Mobile Autonomous Robot. Its goal is to act as a reconnaissance sub-vehicle for the Purdue IEEE Student Branch's entry in the International Aerial Robotics Competition (IARC). The competition consists of four stages:

1) Autonomously fly a 3km course of GPS way-points.
2) Locate a marked building in a group of buildings.
3) Enter the building, locate and photograph a control panel.
4) Complete stages 1-3 in less than 15 minutes.

OMAR's role will be in the third stage entry and reconnaissance of the building. Reconnaissance in this case consists of locating and photographing a control panel on a wall. OMAR will need to autonomously navigate a room, avoid obstacles, and take still images of its surroundings. To perform these tasks IR sensors will be used to do precise room mapping. Sonar sensors will provide object collision detection. Still-image capture will be handled by a camera with VGA resolution to cut down on downstream traffic yet provide a good enough image for processing. The chassis will ride on 4 tires directly driven by independent motors and control will be similar to that of a tank providing 360 degree rotation.

Aside from strictly competing in the IARC, OMAR could also be used for military or law enforcement applications. Autonomous vehicles are becoming ever more popular because it does not have a potential for casualties. They are also a very active field of research even within our own government. The possibilities are nearly limitless and OMAR is part of the new and exciting field of technology and robotics. Shown below are images of the final functional prototype of OMAR along with its block diagram.

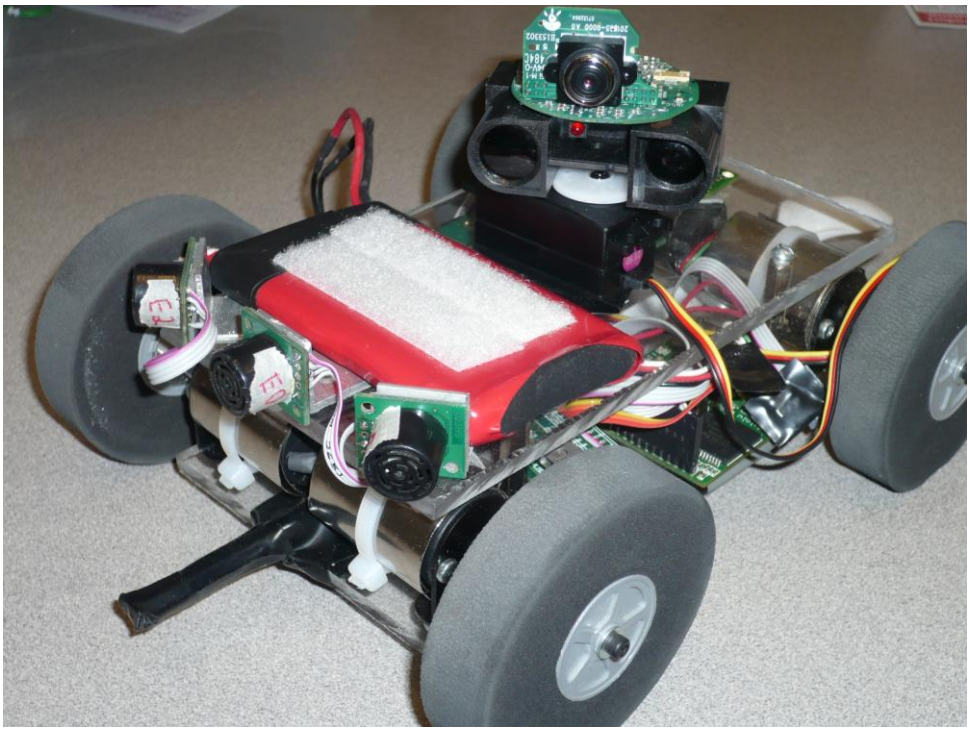


Figure 1 - 1 Final Prototype of OMAR

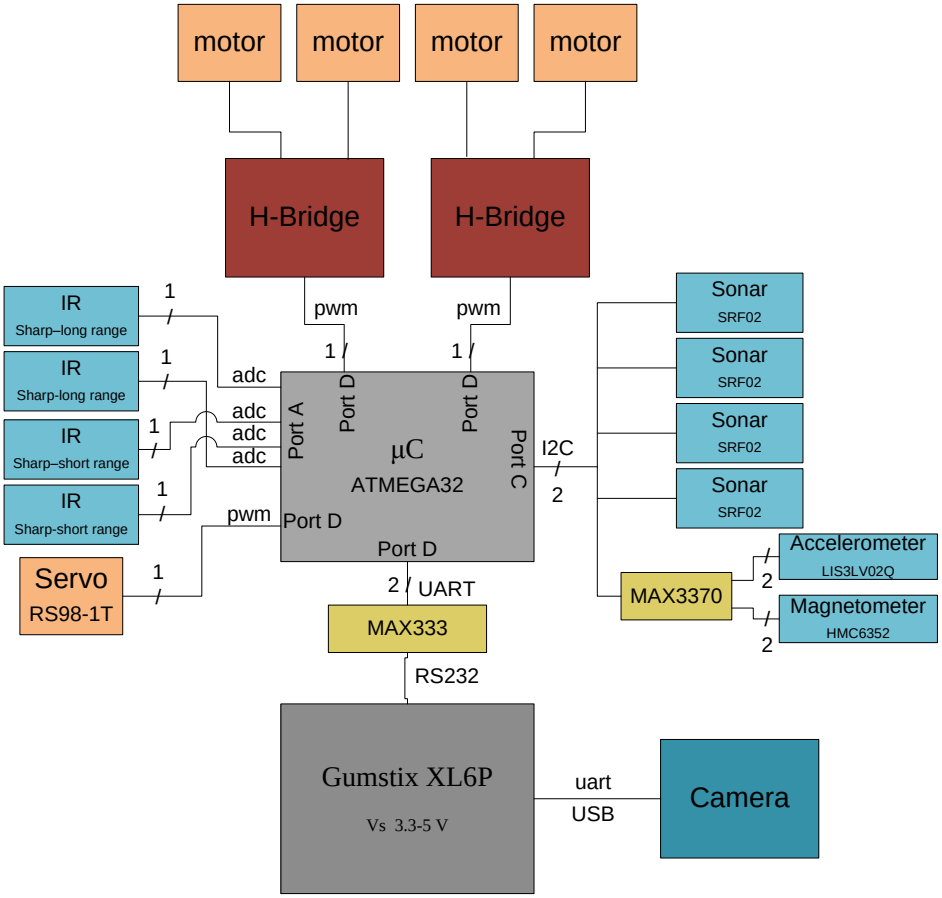


Figure 1-2 Block Diagram of OMAR

2.0 Team Success Criteria and Fulfillment

- *An ability to control vehicle direction and speed*

  OMAR is able to control all aspects of vehicle motion. The direction and speed is dependent upon what the sensors are reading. The speed of OMAR is varied directly proportional to how far an obstacle is; the farther away from OMAR the faster it will drive and vice versa. The direction is changed if the sensors detect an object and depending on which side it is detected on will determine which way to turn.

- *An ability to detect and avoid obstacles*

  Being able to detect and avoid obstacles is an imperative requirement for the autonomous navigation of a room. OMAR has successfully demonstrated these abilities and have used these abilities to control our direction and speed as mentioned above.

- *An ability to capture still images*

  OMAR is able to capture still images. This is done with the Logitech camera that is interfaced with the Gumstix embedded computer via USB.

- *An ability to identify a logo within a captured image*

  OMAR is able to detect faces using the OpenCV library. It is the same software that is used for object detection except that it has already been trained. Training the software with a customized logo was taking a lot longer than the team had anticipated and so the face recognition was employed instead.

- *An ability to autonomously map room and determine vehicle path*

  OMAR was unable to complete this due to insufficient resources. The SLAM algorithm that was going to be used required a lot more memory that was not available on the Gumstix.

3.0 Constraint Analysis and Component Selection

The OMAR reconnaissance vehicle, as complex as it is, has fairly straightforward and obvious restraints that must be addressed in the design process. Seeing as how OMAR will eventually be jettisoned off of a UAV, there has to be size and weight limitations which are acquired from the UAV's physical size and lift capacity. Image recognition and room mapping both require a lot of computational power as well as a fair amount of storage. For the autonomous operation of the rover, it will have to have an array of different sensors that will allow the rover to autonomously navigate the room while avoiding obstacles as well as using these same sensors to map out the room. With all of these sensors, there will have to be a fair amount of I/O on the microcontroller that will interface all of the sensors with the embedded computer. Along with the I/O required for the sensors, it will also require I/O for the motor controllers as well as a link to communicate with the embedded computer. There will also need to be a camera that will be responsible for capturing visual proof of the specified target and then some form of wireless communication to transmit the visual data of the target and the mapped out room.

3.1.1 Computation Requirements

The scope of the competition is perform real time reconnaissance and therefore has a time limit. With the objective of the rover being to identify a logo while avoiding obstacles will require a relatively high amount of computing power. It was determined that the image processing would be the most computationally intensive then followed by the room mapping algorithms and then lastly by the navigational controls. The image processing would have to read in the image which has been determined to be at least VGA quality (640x480) and run image recognition on the images. Photos will also be taken of the main walls and used in the room mapping algorithm. Additionally, the room mapping will have to get information regarding distances from walls as well as the rovers heading from the magnetometer and where it is relative to its starting position. This all has to be complete in a limited amount of time and must process everything real-time as well as send the data wireless via 802.11 or 900 MHz serial transmission back to the base station. The base station does not have any computational requirements because it's only task is to receive information from OMAR. For simplicity, the base station will be a laptop computer.

3.1.2 Interface Requirements

Navigating a room avoiding obstacle will require a plethora of proximity sensors so the rover knows its surroundings. There will be a total of 10 sensors that will detect the surroundings as well as its orientation. In addition, the drive system of the rover will need to be interfaced with the microcontroller. All of the proximity and orientation sensors will also be interfaced with the microcontroller. The camera as well as communication links will be interfaced with the Gumstix computer. Each of the sensors will require different interfacing needs and are summarized in Table 1. Note that some sensors such as the camera will be interfaced directly to the embedded computer while the sensors and motor controllers will all be interfaced to the computer through the microprocessor.

Table 1: Interface Requirements

Usage	Requirement	uC Pins	CPU Pins	Notes
Poll 4 Infrared Range Sensors [2,3]	4x ADC	4	0	10bit precision
Poll 4 SONAR Range Sensors [4,5]	6x I^2C	2*	0	
Poll 1 Digital Compass [6]	1x I^2C	*	0	
Poll 1 3-Axis Accelerometer [7]	1x I^2C	*	0	
Driver 2 H-Bridge Motor controllers [8]	2x PWM	2	0	At least 8bit precision
Position 1 360dg RC Servo [9]	1x PWM	1	0	At least 10bit precision
Comm. between uC and CPU	1x USART	2	2	1Mbaud (May require external xtal to be stable) (Also may need RS232 to TTL level shift via MAX333 if TTL can't be tapped from Gumstix)
Camera	1x USB	0	2	Provided by Gumstix CPU
Debug/Development	1x 10/100 Ethernet	0	120	Proprietary Gumstix breakout interface
Wireless comms to base station	1x 802.11b/g	0	92	Comes attached to ethernet module
	Total	15/11*	214	*I2C pins counted only once since they're shared

3.1.3 On-Chip Peripheral Requirements

All of the sensors needed for room mapping and autonomous navigation have different means of communicating with the microcontroller. There will be four inferred sensors, two long range and two short range, that all require analog to digital converters. The four motors will be driven by 2 motor controllers that will use pulse width modulation. There will also be a turret that will rotate the inferred sensors with a 360° servo that will also require a PWM signal. There will be sonar sensors for the longer distances. There are two sonar sensors that have been picked out with different peripheral interfaces. The SRF02 [5] uses I^2C while the EZ2 [4] uses ADCs. Final decisions on which sonar is used will depend upon accuracy, ease of use, and peripheral requirements and availability. The magnetometer and accelerometer will also require I^2C. Lastly, the microcontroller will have to use UART to communicate with the embedded computer that will have to be level shifted with a max333.

As for the embedded computer, it will need an RS232 port to receive data from the microcontroller. There should be USB capability in the event that a USB compatible camera is used. The rover also needs to communicate with the base station wirelessly. Wireless modules are available for direct connection with the embedded computer along with Ethernet which will be very beneficial for developing on the embedded computer.

3.1.4 Off-Chip Peripheral Requirements

The Gumstix XL6P [10] combined with an ATMega32 [11] can interface all of the required external devices, except for the four brushed DC motors. For this purpose, a pair of ST VNH2SP30-E H-Bridge [8] bi-directional motor drivers will be employed. This part is capable of sourcing a maximum of 30A in both forward and reverse with a V_{CCMAX} of 41V. Each motor requires 2.5-3.0A at the maximum VIN of 12V. Each motor controller will be required to drive two motors in series at VIN of 7.4-11.1V. The constant current supplied by each motor controller should be 4-6A with no more than 20-25A inrush current, which is well within the specifications for the device. The VNH2SP30-E [8] also requires a power MOSFET for reverse voltage protection. The ST STD60NF3LL [12] was spotted in a similar circuit online and meets the requirements for this application so it will be utilized.

3.1.5 Power Constraints

With the intended use of OMAR being autonomous reconnaissance, it should be a standalone system which means it will have to be battery powered. The rules of the competition dictate that the four phases must be completed in less than 15 minutes which has been made the minimum battery life of the rover. There are a lot of electrical components on the rover however many of them use a lot of power except for the motors and the servo. The battery should be able to handle the inrush current of the motors which has been estimated to be no more than 30 amps. The H-bridges [8] and power MOSFETs [12] that will drive the motors will probably need a little heat dissipation which will be accounted for by using heat sinks. It will also be more than likely contained in an open air environment which will aid in the heat dissipation. Most of the devices that are being utilized operate at 5 V however there are a few devices that operate at 3.3 V. The motors chosen operate from between 5 and 12 V. Given these parameters the battery has been chosen to be a 7.4 V battery. The capacity of the battery is still unknown and will depend upon the inrush and continuous current that is pulled by the motors which are estimated to pull a maximum of 8 A total.

3.1.6 Packaging Constraints

The intended use of this rover is to be transported on a UAV and then aerially deployed at a specified location for reconnaissance. Therefore, there will have to be a size and weight restriction as well as surviving up to 3 g's of impact when landing. Even though rigorous mechanical functionality will not be considered in the design of OMAR, the size and weight issues will still be addressed in the design process. It was decided that, considering the lift capacity of the current UAV, the weight would ideally be around 5 lbs. and can be up to a maximum of 10 lbs. The size restriction will have to be relatively small compared to the UAV to ensure that it will actually fit on the UAV and fit well enough so as not to unbalance the UAV while airborne. Obviously, with regards to size, the smaller the better, however the maximum determined size was 14"x6"x6". Another justification for the small size of the rover is it will be traversing an unexplored room and may have small tight spaces to navigate.

3.1.7 Cost Constraints

This project is being funded by the Purdue IEEE Aerial Robotics Committee. Money is not a huge constraint, as their budget is relatively flexible. Regardless, the club has other financial obligations, so the cost should be kept to a minimum. Since compute power is a must, a strong CPU is necessary. The total cost of the chosen computer and essential accessories is near $300, which put a pinch on the rest of the components. As such, costs were cut by building a custom motor controller and using a lesser expensive wireless solution.

The goal of this project, as stated earlier, is to complete stage three of the IARC. This being the case, taking OMAR to the consumer market is not a goal. It would be more at home in a military or law enforcement type market where reconnaissance without risk of human life is vital.

3.2.1 Component Selection Rationale

The most important part of OMAR's success is its main CPU. It must be capable of running intense image detection and mapping algorithms as well as communicate wirelessly with a base station computer. Obviously, high clock rate and memory capacity were a major requirement. But at the same time, the device will be battery powered, so low voltage requirements and an ability to control the clock rate are favorable traits. A USB host controller will be necessary for connecting a camera. Another consideration was size. With the maximum dimensions of around 14"x6"x6" and mechanical components like motors and tires taking up a large portion of this space already, it was important that the electrical components be kept small.

Since the software is going to be fairly complex on its own, an embedded linux solution seemed to be a good choice. Before going to a pre-built computer module, ucLinux was investigated. ucLinux is an open-source project aimed at porting linux to several popular microcontrollers. It has a large user community and supported device list. Unfortunately, their website was very unstable, making research difficult. Even when a device list was obtained, it didn't look like any of the supported microcontrollers had a high enough clock rate, USB host controller and reasonable package. It was during this research that the Marvell XScale PXA series microprocessors were discovered.

The new Marvell XScale PXA series seemed like a perfect solution to the proposed problems. This processor line had clock rates ranging from 600-800MHz and a USB host

controller with one caveat, the only available package is BGA. Fortunately, there are several pre-built computers using this CPU. The two most appealing models were the Strategic-Test TRITON-320 [13] and Gumstix Verdex XL6P [10]. Both have very small footprints and low power requirements.

The TRITON-320 was attractive because of its unique interface with the real world. All that was required was a DIMM200 slot on the PCB to grant access to nearly all of the PXA's peripherals. This module also used the fastest of the PXA series processors (PXA-320) with a maximum clock rate of 806MHz. Another potential problem was finding a small 802.11 card to connect to the PCMCIA bus. Strategic-Test also boasted it as being the lowest power PXA module on the market, using 1.8v SDRAM and NAND flash units. Sadly, after a week of emailing Strategic-Test with no response, and no useful, publicly available datasheet, the search was on again.

Finally, the Gumstix Verdex XL6P [10] was decided upon. Though it has a slightly slower processor and less NAND flash, the expansion boards Gumstix offers were unparalleled. One particular board has 10/100 ethernet, 802.11b/g wireless and an SD card reader. The first two meet some of our requirements, while the third alleviates the lack of flash storage. The XL6P with this breakout board is just under twice the price of the TRITON-320 [13], but development can be done without any more hardware, where the TRITON-320 required a nearly $2000 development board.

Table 2 - Embedded Computer Comparison

	Gumstix Verdex XL6P	**Strategic-Test TRITON-320**
CPU	Marvell PXA-270	Marvell PXA-320
Clock	600MHz	806MHz
SDRAM	128MB	64MB
NAND Flash	32MB	128MB
UART	3	3
USB Host	Y	Y
Ethernet	Y	Y
Wifi	Y	Y (3rd Party over PCMCIA)
SD Card	Y	Y
Supply Voltage	3.6-5.0V	3.3-5.0V
Dimensions	80mm X 20mm	67.6mm X 27.6mm
Breakout	Gumstix Proprietary 60 and 120 pin connectors	DIMM200

4.0 Patent Liability Analysis

In this project, there are two main features that could infringe on active patents. For one case of infringement, a patent would need to claim the ability to autonomous control the body of the robot, and move it around an area while getting input from sensors about the area. This is essential the main idea of the project and thus the most important design feature to look up for infringement. For another case of infringement, a claim would need to specifically discuss how the robot takes in signals from sensors and computes the necessary data for mapping a room, while providing the robot with signals to avoid obstacles in the room. In OMAR's case, a microcontroller takes in all the signals from the sensors and transfers the data to a processing device. That device then maps the room and plots a path for the robot to take. It then sends data back to the microcontroller and tells it how to move around in the area.

4.1 Results of Patent and Product Search

The first patent that closely resembled OMAR was number 6515614 filed on Oct. 11, 2001, titled "Autonomous moving apparatus having obstacle avoidance function."[14] The patent is for an autonomous moving apparatus that moves to a destination while detecting and avoiding an obstacle. It includes devices to detect obstacles and move to a destination under such control as to avoid obstacles. In the first claim for the patent, it describes the comprising parts of the autonomous apparatus. It has a scan-type sensor that scans a horizontal plane to detect a position of an obstacle, while a non-scan type sensor detects an obstacle in the space. It then uses the information from both sensors to estimate the position or area of the obstacle.Using a controller for controlling the apparatus it then travels to the destination while avoiding the obstacle.

The second patent was number 6539284, filed Jul. 25, 2001, titled "Socially interactive autonomous robot."[15] The patent describes that a robot performing substantially autonomous movement will need certain components in order to direct the robot to move within any predetermined safe are, and that it accepts input from a human. The first claim under this patent breaks down the components. It has a processing device, a memory that communicates with the processing device and a mobility structure controlled by the processing device which moves the apparatus. Also, there is at least one sensor for measuring an amount of movement. Lastly, the

memory contains instructions to move the apparatus within a predetermined safe area having a boundary and reference point.

The third patent was number 5170352, filed Aug. 7, 1991, titled "Multi-purpose autonomous vehicle with path plotting."[16] The basic concept of the patent is for an autonomous vehicle to operate in a predetermined work area while avoiding both fixed and moving obstacles. In order for it to accomplish that, it needs a plurality of laser, sonar, and optical sensor to detect targets and obstacles. It then provides signals to processors and controllers which direct the vehicle through a route to a target while avoiding any obstacles. The first claim for this patent is lengthy, and covers a lot of the design for the described vehicle. The claim starts with a vehicle comprised of a body member, wheels, means for propelling and steering the body, and sensors mounted on body. Then it describes three subsystems of the overall design. The first is a machine vision subsystem for receiving the sensor signals and interpreting the signals. Next, a main controller subsystem for receiving input signals. The last one is a navigational subsystem. This subsystem is comprised of a means of receiving input signals from the main controller and machine vision subsystems, and then using those signals to plot a map of the area and a path to a destination. It also sends control signals to control the propelling and steering for the body. It then continuously monitors the signals from the machine vision to determine if an object is in the way or moving.

4.2 Analysis of Patent Liability

From the three patents listed and discussed above, OMAR has literal infringement on all of them. Two of the three only have one feature in their claims that are different from this project; however, those parts are a small part of the whole. The first patent, number 6515614, discussed having a scan and non-scan type sensor. OMAR has both, an IR for scanning, and sonar, for non-scanning. The patent had a detection unit that takes in the sensors data and determines the position of the obstacle, and a controller for moving the apparatus. This is also on OMAR because the microcontroller takes in the sensors data and the embedded computer uses that data to determine the position of the obstacle. The computer then sends signals to the microcontroller to move the wheels. As one can see, OMAR and the claim from the patent are similar and that literal infringement exists.

The second patent, number 5170352, had a lot of components, including three subsystems that worked together. The machine vision subsystem received the sensor signals, the controller subsystem took in input signals, and the navigational subsystem did everything else. The navigational subsystem took in signals from the other two subsystems, and then plotted a map and path for the vehicle. It also continuously monitored signals from the machine vision to determine moving and still obstacles. OMAR covers almost this entire design but in a slightly different way. The project only has two subsystems, the microcontroller, and the embedded computer. The microcontroller takes in the sensors data and sends it to the embedded computer, just like the machine vision and navigational subsystem. The embedded computer will continuously take in the data and map the room, and then send data back to the microcontroller to give it a path to follow, just like the navigational subsystem. The only difference between the patent's claim and OMAR is that the project won't be handling moving obstacles. Its functionality only covers detecting still obstacles. Even with this small difference, literal infringement exists between the patent and the project.

The last patent, number 6539284, provided a basic design for an autonomous apparatus and could move and detect obstacles. It had four main components: a processing device, memory to communicate with the processing device, a mobility structure that moves the apparatus which is controlled by the processing device, and at least one sensor for measuring an amount of movement. This design matches OMAR because the processing device is the embedded computer which already has memory on it. Then the computer gives out commands to move OMAR. Also, the body of OMAR will have an accelerometer to measure movement. The only difference between the two designs is that the patent claims that the memory should have instructions to direct the apparatus to move within a predetermined safe area. OMAR moves in any area, it isn't predetermined. Although this difference occurs between the two designs, there are many similarities that bring literal infringement against OMAR.

4.3 Action Recommended

Since OMAR has literal infringement on all three patents, there are very few options available. The easiest option available is to wait until the patents expire. Even though this option is available, it is basically unusable. Two of the three patents were filled 7-8 years ago which means it would be anther 12-13 more years until those expire. By then, this technology would

most likely be useless anyway. Like stated before, this option is useless. The next option is to change the design to erase the literal infringement. However, this is also useless because everything in the project is needed and there really isn't any way around getting the project done without using what is already being implemented. The next option would involve paying royalties to the owners of the patents. If OMAR would become a commercial product, this would be the only option available to use. However, since OMAR is not going to be a commercial product, and that it is only needed for the aerial project, nothing needs to be done in order to avoid infringing on active patents.

5.0 Reliability and Safety Analysis

As far as safety and reliability are concerned there are many things to consider with OMAR. First, reliability is very critical considering this application will most likely be used for military reconnaissance purposes. If this device were to fail, it may cause the enemy to become aware of the militaries intentions or cost soldiers their lives. OMAR will most likely be very expensive to meet the militaries requirements and because it will most likely only have 1 chance to succeed, it needs to be extremely reliable. Safety is not necessarily a large concern with OMAR. There should be no end user interaction if it is used for its intended purpose. OMAR will be fully autonomous and is not intended to return to the user. Human interaction is only possible in 2 situations. One possibility is with the enemy and safety may not be as big of a concern. The other situation is during testing and safety may be of concern if the user must come in contact with OMAR.

5.1 Reliability Analysis

Omar has quite a few parts that will need reliability and safety consideration. Each part of OMAR is necessary to complete our objective, but not all parts are necessary for partial completion. The magnetometer, accelerometer, and IR sensors are necessary for the room mapping objective, but not for the image recognition. The microcontroller, sonar, and motors, however, are necessary to complete any of the stated 5 PSSC's. Because there are so many components in this project that will need reliability calculations, only 3 components will be chosen as examples.

The 3 devices that we believe are most likely to fail are the voltage regulators, microcontroller, and the motors. All 3 of these components are also critical to the completion of this project. Without one the project would fail. Below are the tables for the reliability calculations for the 3 chosen parts. The microcontroller and the voltage regulator will follow the equation $\lambda_p = (C_1\pi_T + C_2\pi_E)*\pi_Q\pi_L$ for microcontrollers and MOS devices, and the motors will follow the equation $\lambda_p = [t^2/\alpha_B{}^3 + 1/\alpha_W]$ for motors [17]. Tables 1-3 show the selected coefficients and the reasoning behind each choice. The MTTF for the microcontroller is 64.276 years, 29.790 years for the voltage regulator, and 99.13 years for the motors.

Table 1. Microcontroller

λ	1.776 failures/10^6 hours [$\lambda_p = (C_1\pi_T + C_2\pi_E)*\pi_Q\pi_L$] (64.276 years/failure)
C1	.14 (microcontroller)
C2	.015 (SMT)
πT	.84 (80˚C at 8Mhz and 5.0V)
πE	4.0 (ground mobile)
πQ	10.0 (Commercial)
πL	1.0 (> 2 years)

Table 2. Voltage regulator

λ	3.832 Failures/10^6 hours [λ=(C1* πT + C2* πE)* πQ*πL] (29.790 years/failure)
C1	.060 (Linear ~3000 transistors)
C2	.00092 (3 pins SMT)
πT	7.0 (85˚C Linear MOS)
πE	4.0 (ground mobile)
πQ	10.0 (Commercial)
πL	1.0 (> 2 years)

Table 3. Motors

λ	1.515 Failures /10^6 Hours ($\lambda_p = [t^2/\alpha_B{}^3 + 1/\alpha_W]$) (99.13 years/failure)
t	0.0833 hours
α_B	86000 (Motor bearing life @ 30 ˚C)
α_W	6.6e05 (Motor winding life @ 30˚C)

According to the calculations done, the parts that have been chosen for OMAR seem to be extremely reliable. Because OMAR is intended for 1 use only, the MTTF for our parts does not play a significant part, unless it is extremely low (less than 1 per year). Our lowest MTTF is 29.8 years, so OMAR should have no problems with failing irreplaceable parts. One way to improve the reliability of the design would be to simply choose parts with longer MTTF. One software design refinement that would increase the reliability of the design would be to have an initial test ranging devices. The test would simply drive until either the sonar or the IR sensors detect an object and if the 2 sensors are not about the same then 1 of the devices is not working properly. One hardware design change that could be made would be to use a switching power circuit instead of the 5V LDO.

5.2 Failure Mode, Effects, and Criticality Analysis (FMECA)

The criticality levels for OMAR are defined as follows. High criticality is any failure that may cause harm or injury to the user ($\lambda < 10^{-9}$). Medium is any failure that may cause irreplaceable damage and halt the completion of the objective ($\lambda < 10^{-7}$). Low criticality is defined as any failure that causes damage that can be replaced, and does not critically limit the completion of the objective ($\lambda < 10^{-6}$). The schematic for OMAR has been divided into 4 different subsystems: power, software, sensors, and drive.

The first subsystem is the power block and it consists of the battery and voltage regulators as shown in figure A-1. The first possible failure is the battery being shorted. This could be cause by any contact with a current carrying surface. The possible effects of this are the battery exploding. This is a high criticality since it may injury the end user. The next 2 possible failures are due to voltages greater than 5V and less than 3.3 V. Between 3.3V and 5V OMAR would run, except at under 5V some sensors may not function accurately. Incorrect VCC may be

caused by battery failure or LDO failure. This would be considered a medium criticality since it would either damage components, or the objective would not be accomplished. The FMECA table can be found in table B-1 of appendix B.

The second subsystem is the drive subsystem. The schematic is shown in appendix A table A-2. The only 2 failure modes for the motors are the windings and brushes failing. The cause of this could be wear on the brushes or the windings breaking. The effects would be that OMAR would stop moving and this would stop the whole project. Medium criticality would classify these failures because OMAR would not achieve its goal. The FMECA table is located in appendix B table B-2.

The third subsystem is the software subsystem and its supporting components. The schematic is shown in appendix A figure A-3. The main components of the software subsystem are discrete capacitors, resistors, and inductors along with the microcontroller. The first failure is that the microcontroller could lose communication with either the Gumstix or the sensors. This could be caused by either dead ports, incorrect VCC, or failed connections. This would be considered medium criticality since either one would cause OMAR to stop. The only other failure would be the pushbutton failing. This could be caused by any object hit it and causing it to get stuck pressed down. It would affect the microcontroller because it would cause it to shutdown and go into reset mode. This would be a medium criticality since OMAR would again not accomplish its task. The FMECA table for this subsystem can be found in appendix A table A-3.

The fourth and final subsystem is the sensor subsystem. The magnetometer could fail because of either improper VCC or magnetic interference. This would cause the compass heading to be incorrect and OMAR would not drive straight or turn properly. This would be a low criticality since it could still finish its objective, even though it may not drive straight or efficiently. The next failure would be the sonar failing. The sonar could fail do to noise caused on the 40kHz spectrum. This would cause OMAR to not detect objects. High criticality would be assumed since OMAR may run into objects causing possible batter explosion and fire. Next the accelerometer may fail. This could be cause by improper VCC. It would cause the room mapping data to be incorrect as the accelerometer data is used in the SLAM algorithm. Since the image recognition objective could still be accomplished, this would be a low criticality. The last and final possible failure would be if the IR sensors failed. It could be caused by ambient light noise

or improper VCC. The room mapping data would be incorrect as it is used in the SLAM algorithm to determine walls and objects. Again, since the image recognition objective could still be finished, this would be a low criticality.

6.0 Ethical and Environmental Impact Analysis

The environment is a hot topic now more than ever. For far too long has the environmental impact of a product sent to market been short changed, if considered at all. To a similar affect, today's society is rampant with corrupt business men as well as lawsuit happy consumers. For these reasons it is important to consider both the environmental and ethical impact of a new product from concept inception all the way though ship out.

Fortunately, in the case of an autonomous reconnaissance robot like OMAR, the target consumer is looking for more of a research, law enforcement or military solution. This knowledge relaxes the emphasis on the ethical aspects, as the end user's superiors will likely ensure that the safe use guidelines provided with the product are followed and that proper training is received before use. However, it is still pertinent that warnings be given as a reminder of potential hazards, such as contents of hazardous materials and potential for injury.

Environmental considerations are still quite important, especially in military applications where damaged products may simply be left behind. In this situation it is necessary to determine the biodegradability of the chassis, as well as any pollution that may accompany. This can also be applied to devices that are thrown out and end up in landfills. There is sometimes an underlying potential for environmental damage as in the case of lithium polymer batteries. The chemicals themselves are perfectly safe to send to a landfill, but if the pack is sent with too high a charge, there is risk of fire or explosion.

The remainder of this report will outline major environmental and ethical considerations taken during OMAR's design, as well as discuss possible solutions.

6.1 Ethical Impact Analysis

There are ethically few considerations to be made with this project. The biggest problem and the one with the most potential to cause harm to the user, and hence liability to the producers, is improper care and use of the lithium polymer battery pack. Another concern is the

presence of lead in the electrical components. Finally, there is the potential for misuse of the product to invade the privacy of others.

The lithium polymer battery gives an excellent power output to weight ratio making it ideal for this application. However, over charging the pack or handling it roughly can cause a fire or explosion. To make the user aware of these risks, warning labels will be applied liberally. There will at least one in the battery compartment and one on the battery itself. The label will caution the user through words and scary pictures as well as refer to the pages in the manual for proper care and usage of the battery. In the manual, there will be explicit guidelines for charging, discharging and handling the battery safely.

Concerning the presence of lead, the best solution is to simply eliminate it by making the entire design RoHS compliant. Forgoing this though, the only thing that can be done is to design the packaging in a manner as to shield the user from direct contact with the lead bearing components. Also, a warning label should be a fixed in a clearly visible location to make this fact known to the user. It should also be mentioned in the user's manual.

The last issue of misuse by the user to invade the privacy of others really can't be helped. It isn't the place of the producer to tell someone how to use the product, only to give a recommendation. Regardless, there isn't any way to police a policy printed in the manual not to mention the fact that this action would likely insinuate this type of misuse to some users.

6.2 Environmental Impact Analysis

The OMAR prototype is not all that environmentally friendly. The motors and PCB, though the amount is small, both contain lead. This is a problem at all stages of the life cycle. Workers at the factory will have to handle the lead ridden parts as well as breathe air that could potentially contain lead particles from the manufacturing process. In normal use the end user will be handling the device which may have lead residue on it from manufacture, as well as if repairs need made, a technician will likely be touching parts directly containing lead. Disposal of OMAR would contribute to soil and ground water pollution. Another problem is the chassis. The polycarbonate used to make it, though easy to manipulate for prototyping and strong for debugging, is not biodegradable. This is obviously not good for landfills. Lastly, the lithium polymer battery used is actually land fill safe as far as pollution goes. However, if improperly disposed of, the battery could explode or catch fire leading to rather undesirable situations

between the dump point and (or at the) landfill. The remainder of this section will discuss solutions to these problems.

The lead issue can be resolved by pushing for RoHS compliance on all of the electrical parts. This will eliminate hazardous chemicals from all stages of the life cycle. Forgoing RoHS, many steps would have to be taken to protect workers, users and the general public from health risks. These steps would likely cost far more than simply complying with RoHS in the first place. Most all component manufacturers at least offer their devices in an RoHS compliant version. These are often a bit more expensive, but at some point in the future this will likely be a forced standard anyway. The Atmel ATmega32, Gumstix computer, accelerometer module, and magnetometer module are all RoHS compliant already. The discrete components all have RoHS compliant equivalents as well. This leaves only the motors, which are difficult to source with similar specifications for the application and RoHS compliance. These could probably be custom made or made in-house if necessary. Either way, cost would be significantly higher.

The polycarbonate chassis doesn't cause many problems until the disposal stage, though its production does have petroleum components, which increase the carbon footprint. Both of these issues can be resolved by use of newer polymers with better biodegradability properties. There have been advances with these polymers that allow them to be stronger and more durable than before. Some, like the one being developed at Clemson, use a corn byproduct, polylactic acid, in replacement of most of the harmful chemicals currently used in plastic production. [18] Using a material like this wouldn't hurt OMAR's durability but could potentially greatly reduce its impact on landfills as well as litter should it be left on the battlefield.

Lastly, the lithium polymer battery pack. Workers assembling the units would have to be properly trained in handling this type of battery to avoid hazards at the factory. The final packaging should be designed in such a manner as to reduce the possibility of shorting the battery (polarized connectors), as well as minimize the likelihood of the pack being directly impacted in case of a fall or collision. These two steps should lessen the chances of injury to the end users. Chemically, it is safe to dispose of this type of battery in the normal trash. However, if the battery isn't improperly discharged first, shorting the leads or puncturing the pack could result in fire or explosion. Proper instructions for a safe discharge method should be included in

the user's manual as well as a word of encouragement, hinting at the importance of following them.

7.0 Packaging Design Considerations

One of the main concerns for the packing design is that the vehicle will be carried by the helicopter 3 kilometers worth of GPS way points and then be shot into the building. Because of these constraints, the sub-vehicle must be fairly small, as the helicopter that is carrying it is not that large. OMAR must also be very light weight, since the helicopter has a payload of less than 25 pounds. Lastly, OMAR must be very robust, since it must withstand the impact of be shot into the building by the helicopter. Due to that fact that this is an ECE design project, no concern with the operation of insertion into the building will be taken. It will be assumed that the vehicle has landed upright inside the building and proceed from there.

7.1.1 Commercial Product Packaging

There are a very limited number of commercial products that are designed for the intended use of this project. Most of the work being done in this field is for government use. Autonomous navigation, room mapping, and image recognition are all fields of highly active research. When searching for similar products, the vehicles that were found all fell within two intended categories: research or military use. When it comes to the packaging design used in each category, they seem to have some distinct differences. The vehicles for research tend to be much larger in size, more open and less contained, and some were significantly lighter than those intended for military use. The main reasons for this are due to the fact that military UGV's are intended for detection, neutralization, breaching minefields, logistics, fire-fighting, and urban warfare to name a few uses. These applications can require the vehicle to be very small, sometimes heavy, have long battery life, and always very powerful. In research implementation the size and weight of the vehicle are not usually of concern. It is the sensors and software running on the vehicle that is the main concern.

Two main commercial products have been chosen that most similarly resemble the project. The first product, which falls under the research category, is the MobileRobotics incorporation's Pioneer 3-AT [19] with the 2-D/3-D room mapping and stereovision kits. The second product,

the iRobot PackBot [20] with mapping kit, is an example of military use. OMAR is planned to incorporate many features from both products and also have some unique features. Shown below are the advantages and disadvantages of each products packaging design, and also the features of each that OMAR will use.

7.1.2 MobileRobotics Pioneer 3-AT

The Pioneer 3-AT [19] is an example of a UGV that is used primarily in classrooms and for research. The Pioneer 3-AT [19] is capable of autonomous navigation, 2-D room mapping using a laser range finder, 3-D room mapping using stereoscopic vision using 2 cameras, and object avoidance using multiple sonar sensors. The included software allows for the Pioneer 3-AT [19] to perform room mapping straight out of the box. It is quite large, fairly heavy, and does not appear to be extremely robust. It stands at 50cm long x 49cm wide x 26cm tall and weights 12 kg. The design seems to be fully contained with no unnecessarily exposed components.

Fig. 2.1: Mobile Robotics Pioneer 3-AT

The first thing to notice when looking at the Pioneer 3-AT [19] is the huge laser range finder with 2 cameras mounted on top. This laser range finder is placed in the middle of the base and is stationary since the range finder is capable of 180˚ scanning. The range finder is large, heavy, and consumes a large amount of power so it would not be well suited for OMAR. The camera mounted on top is very small, but it houses 2 lenses for stereo vision, which OMAR will not need. OMAR will not include 2 cameras, but the placement of the camera will be replicated, since mounting the camera on top seems to give the best angle for image taking.

The next design specification to note is that the Pioneer 3-AT [19] encloses 8 sonar sensors mounted directly to the front bumper of the base. The placement of this design seems optimal, but the amount of sensors seems excessive. After researching sonar sensors, object detection and avoidance can easily be accomplished by mounted only 2 or 3 sensors on the front at 45˚ or 60˚ respectively. The last main design specification on the Pioneer 3-AT [19] is the drive system. This robot uses a 4 wheel design. This system is cheap and light weight and is very easy to use. OMAR will copy the 4 wheel design.

7.1.3 iRobot PackBot

IRobot [20] offers many UGV's on there website. The PackBot [20] is a standard UGV that can come with many different kits that can perform bomb detection, visual reconnaissance, room mapping, and even sniper detection. The kit that most closely meets OMARs intended application is the PackBot [20] with the room mapping kit. This robot is sometimes used in the classroom or for research, but it is mostly used in the military as are the rest of their robots. This PackBot [20] is capable of room mapping, visual reconnaissance, object detection and avoidance, autonomous navigation, and it can even climb stairs. It is very tightly packed sitting at 20 cm wide x 35cm long x 8 cm tall. The problem with this design is that it weighs 42 pounds! It is heavy, but it is smaller and extremely robust, making more suitable than the Pioneer 3-AT [19] for OMARs design.

Fig. 2.2: iRobot PackBot with mapping kit

The first packaging and design specification to consider is the tank drive system. The tank drive system allows it to disperse its weight across a larger surface area, allowing it to traverse just about any kind of terrain. The triangular tracks in the front allow the tank to climb stairs. This tank drive system with the triangular wheels is more expensive and not necessary for OMAR since OMAR is only mapping the inside of a building and the terrain will not vary.

The rest of the packaging specs on the PackBot [20] are the laser range finder, camera, and sonar sensors. The laser range finder is capable of 360˚ scanning and thus can be mounted anywhere on top of the tank as long as it has clear vision. Again, laser ranger scanners are too large and expensive, so OMAR will not be using one. The idea of scanning 360˚ seems to be more efficient, so this will be incorporated into the design. The single camera is on the front of the vehicle close to the bottom. This seems to be inefficient since the camera will only be able to take pictures from a low angle. OMAR will be using the single camera design mounted. Finally, there are 2 or 3 sonar sensors mounted on the very front of the PackBot [20]. OMAR will copy this design and have 2 or 3 sonar sensors on the front.

7.2 Project Packaging Specifications

As mentioned before, OMAR needs to be very light weight and small. Appendix A contains 3 3-D drawings of OMAR. For OMARs drive system, the 4 wheel design will be used. The left and right sides will be controlled independently so that OMAR can rotate 360˚. On the front of OMAR, 2 or 3 Devantech SRF02 [5] sonar sensors will be mounted at either 45˚ or 60˚ apart. The PCB and Gumstix [10] embedded computer will be place in the middle between the sheets of plastic. On top, the R298-1T [9] servo will be mounted. On the front and back face of the servo will sit the Sharp IR [2] range finders. The reason they will be mounted on the servo is so that OMAR can scan an area of 360˚. In order to help eliminate the IR range finders minimum distance the servo will be mounted in the middle of the vehicle. The body will consist of two sheets of lexan [21]. The motors will be packaged between the 2 sheets of plastic with dimensions 10cm wide x 20cm x 10cm. These measurements were estimated by observing how big the helicopter is and how much room can be used to store OMAR on its landing gear. Foam tires, motors, motor mounts, and mounting hubs from Lynxmotion [22] will be used. The camera will also be place on top of the servo so that OMAR can rotate the camera if needed to take pictures in tight situations. The battery will sit on top of the vehicle located in the back. Finally, the wireless card will be placed on top between the sonar sensors and the servo.

7.3 PCB Footprint Layout

The PCB footprint is shown in appendix C. Listed below in table 4.1 is a list of all the major components and the selected packages. There were not many options for the accelerometer and magnetometer so the listed packages were chosen. The estimated dimensions are 95mm x 95mm and the estimated total area is 9025 mm². All components were placed around the microcontroller based on their interfaces with the microcontroller and where their respective pins are. The dimensions of the PCB were chosen to be as small as possible but still leaving ample room for making all of the necessary traces.

Table 4.1: List of major components and packages selected.

Mfg.	Part	Description	Package	Other options
Atmel	ATmega32	Microcontroller	QFP44	DIP40,QFN44
ST Semi.	VNH2SP30	H-Bridge	SO-30	None
ST Semi.	STD60NF3LL	Pwr MOSFET	DPAK	None
ST Semi.	LIS3LV02DQ	Accelerometer	QFN28	None
Honeywell	HMC6352	Magnetometer	QFN24	None
Maxim IC	MAX1585A	Buck Power Reg	QSOP24	None

8.0 Schematic Design Considerations

There are three main components of the preliminary schematic are power, microcontroller and motor controller circuits. Because of the wide array of sensors that are in use as well as the Gumstix embedded computer [10], ATmega32 microcontroller [11] and the motor controller, the only voltages required are 3.3 and 5 volts. The MAX1858 DC-to-DC step-down converter [23] was chosen because it has two customizable voltage outputs and can handle the power load with ease. The current estimation of our power draw with all of our components connected is just under 9 amps, the bulk of which is the motors. This estimation is based on absolute maximum values given in the documentation for each product employed. Specified in the documentation, a circuit is provided but the values of the discrete components will need to be calculated to accommodate for our specific power needs. This power circuit design was chosen because of its efficiency and relative ease of implementation. It has two sets of circuits that more or less mirror each other to provide the two customizable output levels. The efficiency of the device is due to that of its operation. The two mirrored circuits operate 180 degrees out of phase which allows the ripple voltage and current from the input be reduced. The frequency at which the MOSFETs change state are determined with an external resistor and can be customized between 100 kHz and 600 kHz. The power circuit will be driven by a 7.4 V lithium polymer battery and also has compensation circuitry specified to filter out noise created by the switching of the MOSFETs. Compensation circuitry essentially works as a transconductance error amplifier or a hardware integrator that provides high DC accuracy in addition to filtering out noise.

The motor control will be using two ST H-bridges [8] with a couple power MOSFETs [12] and other discrete components and will be capable of supplying up to 30 A of current. The H-bridges themselves have an operating voltage of 5 V and take three logical inputs that will be provided by general purpose I/O pins. These logical inputs will regulate the direction of current that will be driving the motors and thus the direction that the motors spin. All of the inputs will be optically isolated to ensure the protection of the ATmega32 [11]. The H-Bridges also have a current sensor that is being considered for use however this would have to be read in by an ADC channel and would not be able to be optically isolated. The current that the motors draw will come directly from the battery and will not go through any of the power circuitry and just merely directed by the H-Bridges.

As for all of the sensors, they will all be interfaced with the ATmega32. The ATmega will be running at its native 8 MHz using an internal oscillator. The only two sensors that have an operating voltage of 3.3 V are the accelerometer [7] and the magnetometer [6]. Because of this operating voltage, a level shifter is required. The MAX3370 [24] was chosen to handle the level translation because of its functionality. This device allows for bidirectional level translation which will allow I^2C communication between the ATmega32 and these two sensors. The IR [2][3] and Sonar [5] sensors, H-Bridge, microcontroller and the Gumstix computer all operate off of the 5 V rail from the power supply.

8.1 Hardware Design Narrative

The main subsystems that will be used on the microcontroller will be the ADC, I^2C, PWM and UART. The IR sensors will each use a channel of the ADCs that are located on port A of the ATmega32. There is also the possibility that the current sensors from the motor controllers will be read on the ADC. The rest of the sensors that are being used will be using I^2C that only requires the SCL and SDA pins located on port C of the ATmega32. When communicating with the various devices connected to this port, it will send out the address of which device to communicate with and will do so at 100 kHz. This protocol allows the ATmega to communicate with over 100 devices using seven bit addresses. The 16-bit PWM, with is located on port D will be used to control the servo [9] that will act as a turret that will rotate the camera and IR sensors.

When the servo is not being used the 16-bit timer will be used to create timestamps for any control loops that are being used instead of creating a PWM signal. The other two 8-bit PWMs located on port D and port B will be used to control the motor controller and more specifically the speed at which to drive the motors. Along with the PWM, a few more general purpose I/O pins used as logical outputs and be interpreted by the H-Bridge [8] to determine which way to drive the motors. Lastly, the serial UART output on port D will be used for communication between the ATmega32 [11] and the Gumstix computer [10]. With regards to the Gumstix computer, it too will use a serial interface as well as a USB to communicate with a camera and a wireless interface that will relay information it gathers back to a base station.

9.0 PCB Layout Design Considerations

The main components taken into consideration during the design of the PCB are the microcontroller and interfaces to the sensors, embedded computer, and motor controller. Most of the projects components come prepackaged, like the sensors and motor controller. The distance and proximity sensors need to be placed strategically in order to take reliable data. The orientation sensors need also need to be placed off the PCB so they don't have to be calibrated every time they're used. The motor controller will be placed near the motors and doesn't need to be on the PCB. The last part is the embedded computer, which is also off-board since there isn't a way to connect it to the PCB. At the same time, having that with the camera and wireless link will take up a lot of room and doesn't need to be on the board. Since all of those pieces are already designed and on boards already, the PCB just needs headers to connect those components to the microcontroller.

The board can be broken up to sections covering power, digital, and analog. These three sections need to be separated in order to minimize EMI throughout the board [25]. The power section covers the regulators that convert the incoming battery voltage down to 5.0 and 3.3 volts. Power traces will be 70 mil since ample space exists, which will help make the power lines less resistant [25]. The digital section covers the microcontroller and most of the traces to the headers. These traces will be 12 mil and the headers will be placed all around the microcontroller to shorten path lengths to limit the chance of EMI [25]. Two level translators are necessary for the accelerometer and magnetometer since they run at 3.3 volts while the other

sensors run at 5.0 volts. Those two components have bypass capacitors for each of the two power signals coming in, which is recommended by the manufacturer to be 0.1uF per capacitor [24].

The analog section covers the ADC lines that go from the microcontroller to the IR sensors and motor controller. These lines need to be clear from power lines to reduce noise and interference [25]. This will be accomplished by placing nothing else around or routing nothing else through that area.

9.1 PCB Layout Design Considerations - Microcontroller

The microcontroller has three pins for power and another line to power the ADC converter. Bypass capacitors placed between power and ground will help reduce the load on power lines and remove unwanted glitches. The manufacturer advises to put a LC circuit on the power for the ADC converter to help reduce noise [11]. The capacitors and inductor need to be placed as close to the microcontroller to reduce noise on the power lines, and since they are small enough to be surface mounts, they can go on the back of the board, under the microcontroller [25]. Also, to avoid any disturbance on the ADC inputs, power traces and other components will not be placed near those lines. One of the most critical traces going into the microcontroller for the reset pin, so the trace will be placed so that little noise and interference could cause it to jump, causing the microcontroller to behave erratically [25]. Also, the reset pin has a resistor and a pushbutton that stabilizes that trace and pin.

9.2 PCB Layout Design Considerations - Power Supply

To provide the right amount of voltage and enough current to all parts on and off the PCB, three regulators will be used to provide two 5.0 V rails and one 3.3 V rail. The reason behind utilizing three regulators is to be able to meet the requirements for the current draw. Almost every component in the design ranging from the embedded computer, microcontroller, motors, and to almost all the sensors run on 5.0 volts, while the accelerometer and magnetometer run on 3.3 volts. The reason for two sources of 5.0 volts was to split up everything on the board and the motors from the embedded computer. The embedded computer will also use a USB connection for a camera and a wireless connection, which in this case, causes this part of the circuit to require more than 3.0 amps of current. The regulator for the embedded computer will

be able to handle up to 7.5 amps, while the regulator for the rest of the components will be 3.0 amps. The 5.0 volt regulator at 7.5 amps needs a 10uF capacitor on the input, and a 150uF capacitor on the output, which was specified in the datasheet [26]. For the 5.0 volt regulator at 3.0 amps, the manufacturer recommends that a 0.33uF capacitor should go on the input and a 0.1uF capacitor should go on the output as bypass capacitors to help reduce noise from propagating throughout the circuit [27]. The 3.3 volt regulator needs to have a 0.47uF capacitor along with a 33uF capacitor [28].

10.0 Software Design Considerations

Several aspects must be considered when designing the software for an autonomous vehicle like OMAR. Sensors must be read consistently and at an appropriate rate. These values need to be filtered, integrated with the current state and control signals modified as necessary. It is important that all of these things happen in a timely manner and that data from the sensors be received on time. A real-time system is an excellent solution to all of these problems. Its main goal is to meet deadlines, making it the perfect development paradigm for OMAR.

10.1 Software Design Considerations

To solve the real-time problem an embedded Linux operating system was chosen. With the addition of a preempting process scheduler that operates in O(1) time complexity added in kernel version 2.6 as well as a high resolution timers, priority inheritance and generic interrupt layer enhancements in version 2.6.18 Linux is considered to be a “soft” real-time system. [29], [30] To take advantage of the process scheduler, the software will need to be extensively threaded and its niceness needs set low to give it high priority. The software running on the Gumstix embedded computer will need to perform a couple of compute heavy tasks (mapping and object detection, service several potentially high latency devices (network, USB and UART) as well as respond quickly to requests from the microcontroller. To ensure all of this work goes on harmoniously, the application will be event driven using callbacks and heavily threaded. C++ is the chosen language for its encapsulation and data protection properties. The POSIX threading library is employed for parallelization and synchronization. Development takes place on Linux work stations using a GNU arm-linux toolchain provided by Gumstix.

The motor controller and servo, are both utilizing the timers that are set for fast PWM mode. The fast PWM mode was chosen because it uses a single slope operation which provides twice the frequency capabilities of the phase correct PWM mode. The frequency of the PWM was chosen to be 3.9 kHz because the maximum frequency that the motor controller is 20 kHz and 3.9 kHz worked out best for the pre-scaling options from the system clock. The timer also needs to be initialized, along with the frequency, to the asserted output compare state. The duty cycle of the asserted stated can then be controlled using the output compare register which will change continuously to help OMAR driving straight with the assistance of the magnetometer. The motor controller also requires four GPIO pins, two for each channel, to indicate which direction to drive the motors. These GPIO pins are located on port C which is also utilized by the JTAG programming interface. The JTAG interface needs to be disabled, because it uses four needed pins. This is done by programming the fuse bits of the microcontroller.

The magnetometer, accelerometer, and sonar sensors are communicating on the i2c bus. The i2c line will be initialized to run at 100 kHz standard speed. This speed was chosen because each device recommended running in standard mode (100 kHz). The IR sensors output an analog signal so they will be monitored on the ADC. The ADC is set to not run in interrupt mode. The communication between the Gumstix and the ATmega32 will be over the UART. The UART will run at 38400 BUAD. This is because at 8 MHz the maximum speed with the smallest percentage of data is 38400 BUAD. Because the sensors on the i2c bus do not have interrupt pins, the ATmega32 will operate in a polling loop. Each loop will collect all sensor data and send it to the Gumstix.

10.2 Software Design Narrative

10.2.1 Gumstix

The Gumstix software will consist of a main control thread which will have the task of spawning the worker threads, registering callbacks and cleaning up after the workers once they have terminated. It will also stop the vehicle if it is determined that the mission is complete. CPU usage by this thread will be relatively low as it will spend a majority of the time asleep.

Two threads will be created to do heavy computation. One will handle image processing using the object detection functions of Intel's open source computer vision library **OpenCV.** When it finds the target object it will callback to the main control thread which will

stop the vehicle and send the image back to the base station. The other CPU intense thread will take care of mapping using a SLAM (Simultaneous Localization and Mapping) algorithm. Two potential candidates for this algorithm are being researched, GridSLAM and DP-SLAM, both of which are hosted at **OpenSLAM.org.** This thread will maintain a map of what has been "seen" so far in the room. The map will be used to determine where to explore next, by determining the largest opening in the map. When it is decided that the room has been sufficiently mapped (a large percentage of the map is closed), the thread will signal the control thread to stop the vehicle as the target image is not likely in the room.

Three more threads will be spawned to handle the network, camera and telemetry data. The networking thread will run a server using basic TCP socket programming. The base station client will connect to it over an AdHoc 802.11b/g network. Primarily, this connection will be used to send the target image back to the base station once found. If necessary, its function will be extended to provide debugging and control to the vehicle. The camera thread will take a picture with the USB webcam and signal the image recognition thread for processing when it has finished. The camera takes ~20ms to take a picture giving way to ~50 fps, which is definitely overkill for this application. The frame rate will likely be limited to 3-4 fps. The last thread will communicate with the microcontroller over the UART. Primarily, it will receive telemetry data to be sent off to the SLAM thread, but will also be used to start and stop the vehicle as well as transmit new headings.

All portions of the Gumstix codes have been diagrammed. Any non-trivial code segments have been laid out in pseudocode. The UART class is mostly complete. A test stub for the camera has been written for x86 which also compiles without error under ARM, the kernel driver needs to be compiled before it can be tested.

10.2.2 ATMega32

The only module that the motor controller and servo utilizes is the PWM. Since the PWM is in the asserted output compare state, the output compare register will control how much of the duty cycle will be asserted as opposed to the other way around. The H-bridges located on the motor controller determine the speed to drive the motors from how long the PWM cycle is asserted. The H-bridges also have two GPIO pins per channel which determine the state of the motors to drive forward, reverse, brake to ground, or brake to V-. The code for this part of the

project is completed entirely to allow the car to drive, however testing needs to be done to see how straight OMAR will drive. Assuming that it will not drive straight, the magnetometer will be implemented into the drive forward and reverse functions that will vary the PWM duty cycles to compensate to help OMAR drive straight. The magnetometer will also be used to turn OMAR and the motor controller will drive each side of wheels in opposite directions until the desired angle is reached. Neither of these considerations has been completed and will be once the frame is built and OMAR is mobile.

The magnetometer, accelerometer, and sonar sensors all use the i2c module. For each device, the ATmega32 will send the address of the device it wants to read, send the command to acquire data, and then continue to read out the necessary number of bytes. For the IR sensors, the ADC is triggered to take a sample and then the data is copied to a variable. Once all data is collected, the main loop will dump the data to Gumstix via the UART. The functions for each of these devices has been written and tested on the PCB. The micro can successfully read each sensor and send the data over the UART. The Accelerometer will aid in knowing how far we have moved and is essential for the SLAM algorithm. The Sonar sensors will be used for object avoidance. The IR sensors will be used for room mapping and is also essential for the SLAM algorithm. The only thing that is left to be done is that each of the sonar devices needs to have a different address, so we will have to change the addresses of 3 of our sonar devices.

11.0 Version 2 Changes

Looking back on this project there are a few changes that would have been made if a second chance was given. The first thing to change would be to make the frame more robust in order to withstand being launched from the helicopter. The frame would also be modified so that is was a closed shell of some sort to make it more stable, to protect the sensors, and to make it more aesthetically pleasing. The IR sensors were not very reliable and did seem to provide accurate data. Instead, an expensive laser scanner, sonar sensors, or a much more improved ADC noise canceling circuit would be used. A switching power supply would be used as it is more reliable. Also, a regulator that can supply higher currents would be used as the Gumstix seems to draw much more current than expected. Along those lines, a battery charging circuit would be very beneficial so the battery does not have to be unplugged to be charged. The solutions for implementing SLAM were to computationally intensive, so alternate solutions would be pursued.

Finally, the internals of OMAR are very hard to connect and fit everything together nice and compact, so a better method of cable management and part placement might be considered.

12.0 Summary and Conclusions

This entire project has been a huge accomplishment for all of us. This is the first time most of us have ever had to deal with a project that consisted of both analog and digital hardware as well as software. There are no clear guidelines on how to accomplish the tasks, you have to research and read up on how to get it done. We were able to successfully take a popular hobby microcontroller and exploit almost all of its interfaces. We were able to interface with various sensors given little documentation. This was also our first time working in a real time environment, so our Gumstix code was heavily threaded. Two PCBs were made where the latter of the two had better noise suppression and was successfully implemented in the design.

Throughout this entire project a lot of lessons have been learned. The I2C bus is heavily dependent on timing and delays. It was learned that one must take care in making sure that the delays and timing are setup properly. Sonar devices are only accurate if the object it is facing is square with the sensor, otherwise the sound waves bounce off in the wrong direction. The microcontroller can be locked if you are not careful when setting the fuse bits, which was experienced when setting the internal oscillator to 8Mhz. Wiring up the PCB, and the placement of devices are also important. The ADC requires a lot of different noise canceling techniques in order to eliminate enough noise to reliably read the IR sensors. We also learned that open source SLAM algorithms require around 8 CPU's and a large amount of ram, which is not very well suited for a mobile embedded environment.

13.0 References

[1] Purdue IEEE, “Purdue IEEE Student Branch,” Purdue IEEE January, 2008. [online]. Available: http://purdueieee.org . [Accessed: February 6, 2008].

[2] Sharp, "GP2Y0A02YK," [Online Document], unknown publication date, [cited January 31, 2008], http://www.acroname.com/robotics/parts/gp2y0a02_e.pdf .

[3] Sharp, "GP2Y0A700K0F", [Online Document], 2005 August, [cited January 31, 2008], http://www.acroname.com/robotics/parts/R302-GP2Y0A700K0F.pdf .

[4] MaxBotix, “LV-MaxSonar®-EZ2™ High Performance Sonar Range Finder,” [Online Document], 2007 January, [cited January 31, 2008], http://www.maxbotix.com/uploads/LV-MaxSonar-EZ2-Datasheet.pdf .

[5] Acroname Robotics, "Devantech SRF02 Sensor," Acroname Robotics, December, 2007. [Online]. Available: http://www.acroname.com/robotics/parts/R287-SRF02.html . [Accessed: Jan. 31, 2008].

[6] HoneyWell, "Digital Compass Solution HMC6352," [Online Document], 2006 January, [January 31, 2008], http://www.sparkfun.com/datasheets/Components/HMC6352.pdf .

[7] STMicroelectronics, “LIS3LV02DQ”, [Online Document], 2005 October, [January 31, 2008], http://www.sparkfun.com/datasheets/IC/LIS3LV02DQ.pdf .

[8] STMicroelectronics, “VNH2SP30-E,” [Online Document], 2007 May, [cited January 31, 2008], http://www.st.com/stonline/products/literature/ds/10832/vnh2sp30.pdf .

[9] Acroname Robotics, " High Torque Full Turn Servo," Acroname Robotics, January, 2008. [Online]. Available: http://www.acroname.com/robotics/parts/R298-1T-SERVO.html . [Accessed: Jan. 31, 2008].

[10] Gumstix, "Specifications of the Gumstix Verdex Motherboards," DocWiki, December, 2007. [Online]. Available: http://docwiki.gumstix.org/Verdex . [Accessed: Jan. 31, 2008].

[11] Atmel, "ATmega32," [Online Document], 2007 August, [cited January 31, 2008], http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf

[12] STMicroelectronics, “STD60NF3LL,” [Online Document], 2006 July, [cited January 31, 2008], http://www.st.com/stonline/products/literature/ds/7680/std60nf3ll.pdf .

[13] Strategic Test Corp, "TRITON-320 PXA320 module," [Online Document], unknown publication data, [cited January 31, 2008], http://www.strategic-embedded.com/pxa320_monahan_linux_wince/datasheet_triton-320.html .

[14] Google, "Autonomous moving apparatus having obstacle avoidance function," Google. [Online]. Available: http://www.google.com/patents?id=2IYOAAAAEBAJ&dq=6515614. [Accessed: Mar. 25, 2008].

[15] Google, "Socially interactive autonomous robot," Google. [Online]. Available: http://www.google.com/patents?id=ng0PAAAAEBAJ&dq=6539284. [Accessed: Mar. 25, 2008].

[16] Google, "Multi-purpose autonomous vehicle with path plotting," Google. [Online]. Available: http://www.google.com/patents?id=DlEgAAAAEBAJ&dq=5170352. [Accessed: Mar. 25, 2008].

[17] Department of Defense, "Military Handbook Reliablity Prediction of Electronic Equipment", [Online Document], Janurary 1990, Available HTTP: **http://cobweb.ecn.purdue.edu/~dsml/ece477/Homework/Fall2006/Mil-Hdbk-217F.pdf**

[18] "New Corn-Based Plastics Considered Durable, Biodegradable" Aug. 23, 2004. http://usinfo.state.gov/gi/Archive/2004/Aug/24-641357.html

[19] MobileRobots inc., "Pioneer 3-AT with room mapping kit," MobileRobots inc. January, 2008. [online]. Available: http://www.activrobots.com/ROBOTS/systems.html . [Accessed: February 6, 2008],

[20] iRobot, "PackBot with mapping kit," [online document], unknown publication date, [cited February 6, 2008], http://www.irobot.com/pass.cfm?li=filelibrary/GIspecsheets/MappingKit.pdf

[21] Professional Plastics, "Lexan Sheet," Professional Plastics. [Online]. Available: http://www.professionalplastics.com/LEXANSHEET9034 . [Accessed: Feb. 06, 2008].

[22] Lynxmotion, "Lynxmotion Robot Kits," Lynxmotion.[Online]. Available: http://www.lynxmotion.com . [Accessed: Feb. 06, 2008].

[23] Maxim-IC, "MAX1858," [Online Document], 2003 October, [cited Feb. 14, 2008], http://datasheets.maxim-ic.com/en/ds/MAX1858A-MAX1876A.pdf .

[24] Maxim-IC, "MAX3370," [Online Document], 2006 December, [cited Feb. 14, 2008], http://datasheets.maxim-ic.com/en/ds/MAX3370-MAX3371.pdf .

[25] Motorola, "System Design and Layout Techniques for Noise Reduction in MCU-Based System," [Online Document], 1995, [cited February 20, 2008], http://cobweb.ecn.purdue.edu/~dsml/ece477/Homework/CommonRefs/AN1259.pdf

[26] Linear Technology, "LT1083," [Online Document], [cited April 24, 2008], http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1003,C1040,C1055,P1283,D3741

[27] Fairchild, "LM78XX: 3-Terminal 1A Positive Volage Regulator," [Online Document], 2006 May, [cited February 20, 2008], http://www.fairchildsemi.com/ds/LM/LM7805.pdf

[28] National Semiconductor, "LM3940," [Online Document], 2007 July, [cited February 20, 2008], http://www.national.com/ds.cgi/LM/LM3940.pdf

[29] Jones, M. Tim. "Inside the Linux Scheduler," June 30, 2006, http://www-128.ibm.com/developerworks/linux/library/l-scheduler/?ca=dgr-lnxw09LinuxScheduler

[30] "Linux Kernel Gains New Real-Time Support," Oct. 12, 2006, http://www.linuxdevices.com/news/NS9566944929.html

Appendix A: Individual Contributions

A.1 Contributions of Michael Cianciarulo:

During the design stage of the project, I spent some time researching into range finder devices. There were a few options that I looked into including: IR, sonar, laser, and stereoscopy. I discovered a lot of projects used laser because it gives you the best measurement. However, after looking into it further, I discovered that any laser device available was way too heavy, and the cost was extremely high. As another teammate looked into IR, I spent time with sonar and found a relatively cheap and accurate sonar. The next parts that I researched for were batteries, motors, and a wireless module. During the end stage of researching into parts needed for the project, I started a table on a white board with a list of all parts needed. Then I filled in the part name, number, power and current requirements, and type of interface for each one.

I helped work on the schematic which was a stepping stone into my next job, the PCB. Initially, we were going to have a power supply circuit, so I did the whole schematic for that. Before doing the PCB, I had to go back in and change up the microcontroller schematic. I added in headers for all the devices to connect to the board. After looking into datasheets for various parts, I added bypass capacitors based on the manufactures advice. During the time I was working on the schematic, I was also experimenting with Orcad Layout since I had never worked with that software before. After playing around and testing the software, I imported the schematic. It took a while to look up all the footprints for all the parts for the board. I also had to make footprints for the level translators, barrel jack, and the pushbutton. During this time some parts of the schematic were changing, like more pins for some headers, or more discrete components. Then I would have to update the PCB at the same time. I also read into PCB design to see how to trace power lines, where to put the ground pour, and how to design the board to reduce EMI.

After receiving the initial PCB, a week or so, I started designing a new one. Some mistakes were made on the first board. For example, the footprint for the pushbutton was wrong. Also, the LED and a capacitor were put on the bottom of the board, and we wanted them on the top. Also more headers were added so every pin of the microcontroller was available for use.

I also helped out in a few other parts of the project. I helped look into SLAM for the mapping part of the project. Some papers were available that explained all about it. I also looked

into different sets of code available. The next job was to design a separate power board because the 5.0V regulator wasn't going to be able to supply enough current to the whole board. We needed a separate one for the embedded computer. Instead of doing a new PCB, we decided to just have a small separate board. I did the initial work for that board by drawing up the circuit and getting the needed parts for it.

A.2 Contributions of Josh Wildey:

Josh Wildey was the only EE on the team, so he mainly worked with the circuitry and the hardware. He made all of made all of the initial schematics with the help from Mike since he was the one making the PCB. Initially, a switching power supply was going to be used, but it was later decided to use LDOs because they were much simpler and left little room for error. The motor controller came was bought to be able to get OMAR moving before the PCB was made.

Josh also made the initial chassis for OMAR to test the motor controller. The chassis is very similar to that in Appendix B with minor fluctuations in the measurements. Once there was a platform to test on, he wrote code for the microcontroller to interface it with the motor controller. The motor controller needed a total of 6 inputs from the microcontroller. Two of them were the PWM signals that came from the 8-bit timers, and the other four were GPIOs from the ATmega which specified direction. The JTAG programming interface had to be disabled to use port C for GPIO pins for the motor controller.

After getting OMAR moving, Josh started looking over the code that RJ had written that interfaced all of the sensors and the communication code. After getting familiar with RJ's code, he wrote a proportional-derivative loop that would be used to make OMAR drive straight by varying the speed of the motors. Initial testing was done with the loop without actually driving and just testing the values received from the PD loop. After testing was done along with playing with the gains of the loop, the magnetometer died, and was left out of the design because of time constraints. Josh then started to help out with the state machine for the main loop for the microcontroller. RJ broke off from doing this and started to work on the image processing aspect of the project. OMAR, at this point, was detecting objects fairly well except for chair and table legs as well as walls when it approached them at a certain angle. The sonar sensors were repositioned and thresholds were changed to fix this problem.

A.3 Contributions of Trent Nelson:

Trent Nelson's voluntary responsibility was software, in particular, the software for the Gumstix minicomputer. The team decided early on that the Gumstix software need to as real-time as possible, which meant excessive threading. More requirements came to follow including TCP network communication with the base station, serial communication with the microcontroller and USB communication with a webcam. Trent had the most experience in these areas, so it made sense that he take responsibility for this aspect of the project. Though this was his main focus, he also helped out with the other aspects including overall conceptualization and design, debugging, software for the microcontroller, debugging, PCB design, mechanical design and debugging.

The group worked with Trent's initial vision to design the physical chassis and drivetrain, modifying it where need be, resulting in the final product. He then went on to research the Gumstix development environment and open source libraries to aid in the the more difficult tasks of the software. During the research phase Trent found the drivetrain components for OMAR from Lynxmotion (motors, motor mounts, wheels and hubs), as well as the Pololu motor contoroller. He also looked into a different minicomputer, also based on the Marvel Xscale microprocessor, which was replaced by the Gumstix due to lacking customer support. Trent helped RJ debug his sensor test code as well as with controls in the final software. He helped Josh brush up on his C skills so he code hash out some code for the motor controller. Lastly, he helped Mike create some custom layouts for the barrel jack and I2C level translators, as well as some miscellaneous layout considerations for the PCB.

Trent spent the majority of his time on the Gumstix software. The programming started with small tests of the Gumstix ARM tool chain to get a feel for its capabilities. The consensus was that it was close enough to x86 to do a majority of development on a Linux desktop system, which worked well with some Makefile magic for an ARM build. C++ was chosen to make use of the object oriented programming it provides. The POSIX threads library (libpthread) was employed for its robustness and portability. Intel's OpenCV library handled the image detection algorithm and is open source. The SLAM mapping and localization ended up being scrapped as for it to be in any way accurate required large amounts of memory and grid

computing, far from real-time and nowhere near deployable on a robot of OMAR's size. The rest of the code came from the standard C libraries.

Trent also had some non-technical contributions. He wrote the Software Design Analysis paper as well as the Environmental and Ethical Considerations paper. He also setup an SVN repository on a Purdue IEEE Student Branch server for concurrent versioning of all of the software projects. To complement this he setup an on-line source browser using viewvc that allowed for viewing of source code with syntax highlighting and version differences all from the comfort of a web browser.

A.4 Contributions of Robert Toepfer:

During the conceptual design stage of the project Robert was responsible for a few different things. First, Robert found a decent web template to use for our website and laid out the basic structure for our website. Aside from the basic setup of the website, Robert researched many of the different components to be used. Robert first looked into the various options for a high speed embedded computer to do the image recognition and room mapping. For this application, X-scale processors were considered since they had much higher clock speeds than most embedded systems that were not extremely expensive. Initially, the Triton320 was considered, but it was eventually determined that a Gumstix module would be better to use due to its packaging, documentation, and prior experience. Robert also researched some methods for distance measuring and room mapping. Robert decided that IR and sonar sensors would be the best and options for distance ranging due to the low cost and range. Laser range finders would be more ideal, but they are heavy, large, and expensive. Robert selected the Sharp IR sensors and Devantech srf02 sonar sensors for the distance ranging. After researching online, Robert discovered that SLAM was a common technique used for 2-D room mapping. A little research was done reading up on SLAM and the ideas behind it. It initially seemed like SLAM was the way to go, and there were open source implementations online ready to use.

After all the parts were determined and purchased, Robert was responsible for most of the microcontroller and sensor work. Robert first started out by initiating the UART and communication with the micro via serial to a computer. Robert wrote all of the I2C drivers which

our sonar, IR, magnetometer, and accelerometer sensors used. The code for each I2C device ended up having to be different. For the sonar sensors, you had to send the address, the register to read or write, and then the command. For the magnetometer, you sent the address and the data to write, or the address followed by an extra acknowledgement to read the 2 bytes of data. For the accelerometer you had to send it the address, and each successive register to read. The accelerometer returns 6 bytes of data (2 bytes per axis) so you had to send it 6 register locations in a row. After realizing that the accelerometer and magnetometer operated at 3.3V Robert ordered the level shifters and had Mike add it to the PCB. One of the main problems Robert had with the I2C devices was that they were all timing sensitive. Robert originally did not have the F_CPU setting to the correct frequency, so all of his delays were off. This caused all the devices to operate sporadically. Robert also wrote the code for the IR sensors (ADC), but the IR sensors were eventually not used.

When the first PCB arrived, Robert helped soldered most of the discrete components on the board. After the PCB was ready, Robert started the initial testing of the sonar sensors. Robert tested different arrangements of sensors and the number of sensors, as well as varying different settings in the code such as motor throttling and object distance thresholds. Robert worked with the PID code and the magnetometer in order to help OMAR drive straighter, but that ended up not being used because the team managed to break two magnetometers. The next thing Robert did was change the I2C code so that each device was “pulled” so that OMAR did not have to wait for the timing constraints of each device. This was necessary due to the real time constraints of the project. The sonar test code was later revamped to include a sonar data structure and later became the main microcontroller loop.

Lastly, Robert wrote the functions used to communicate with the Gumstix. Throughout the whole project every member of the team contributed in creating headers, wires, re-soldering parts, etc. Robert helped with placement of different parts on OMAR. The serial level translator ended up being soldered incorrectly, so Robert fixed that along with the new power board for the Gumstix (which was initially incorrect as well). Robert also worked with the Open-CV image recognition software to train the image for recognition. It went well, except the haartraining.exe stalled at stage 17 of the training. From past user’s experience, a training of 20 stages or more is needed for accurate recognition. Robert also researched how to use the XML files for image recognition.

Appendix B: Packaging

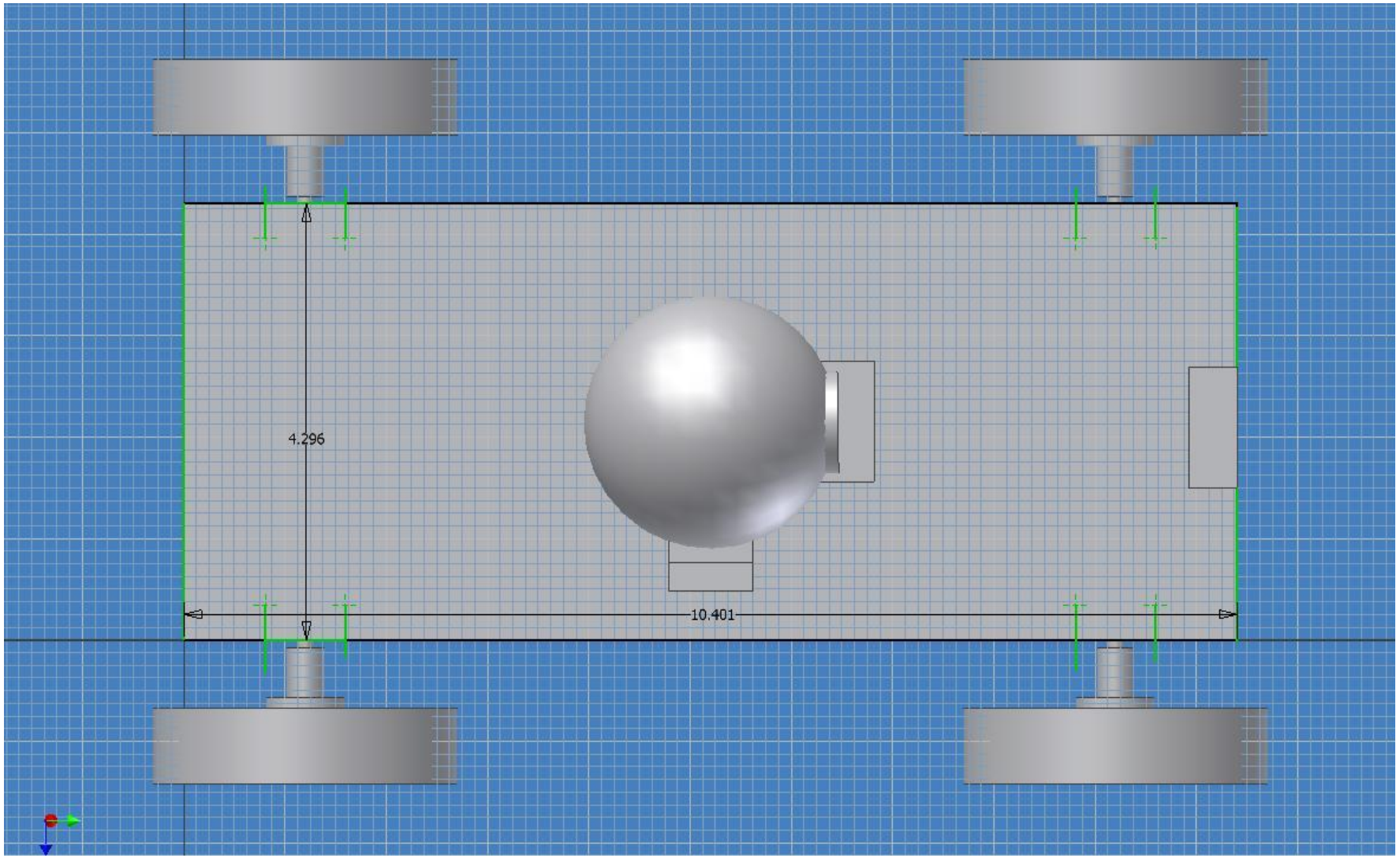


Figure B-1 Top View of Conceptual Packaging of OMAR

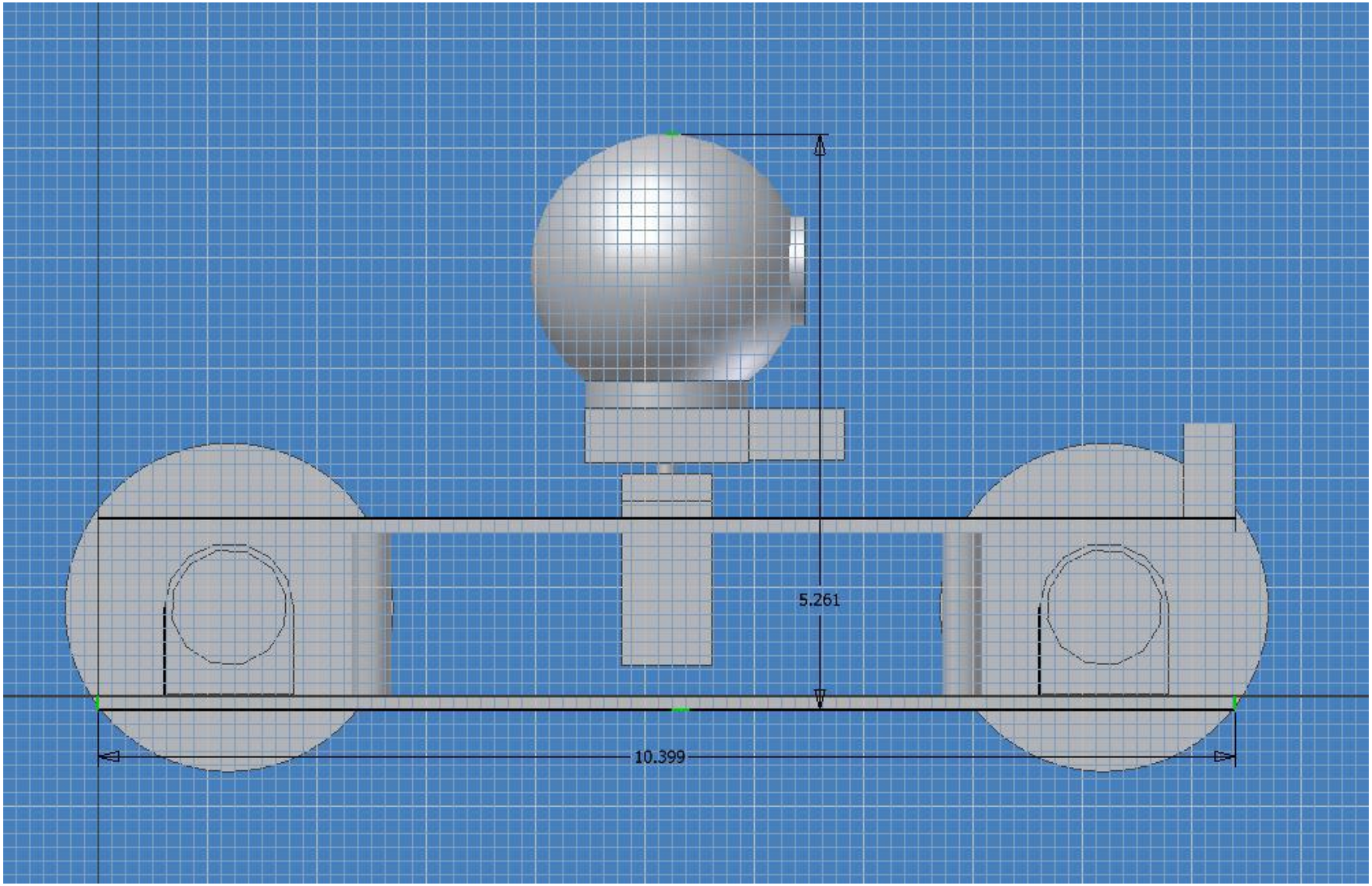


Figure B-2 Side View of Conceptual Packaging of OMAR

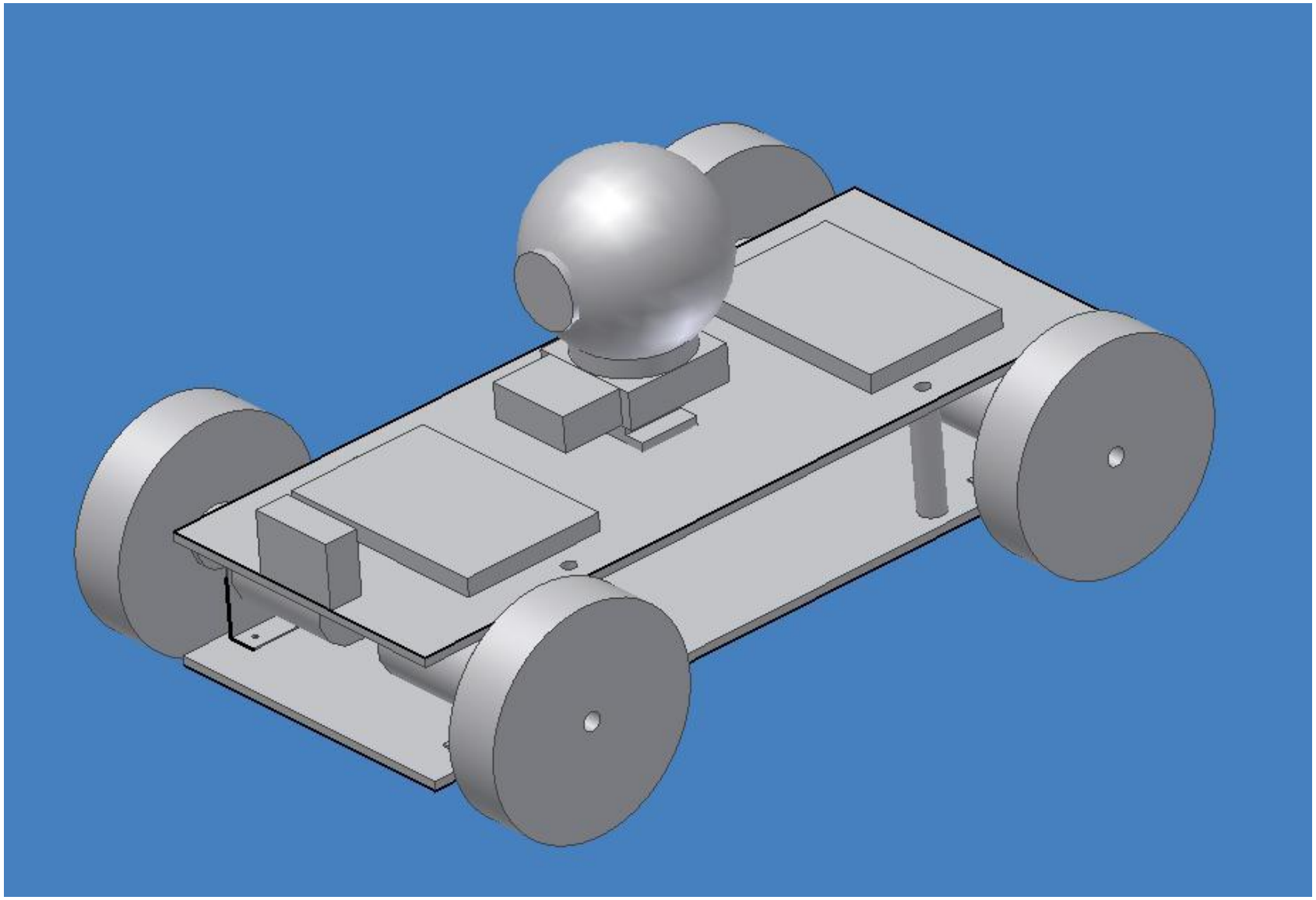


Figure B-3 Top View of Conceptual Packaging of OMAR

Appendix C: Schematic

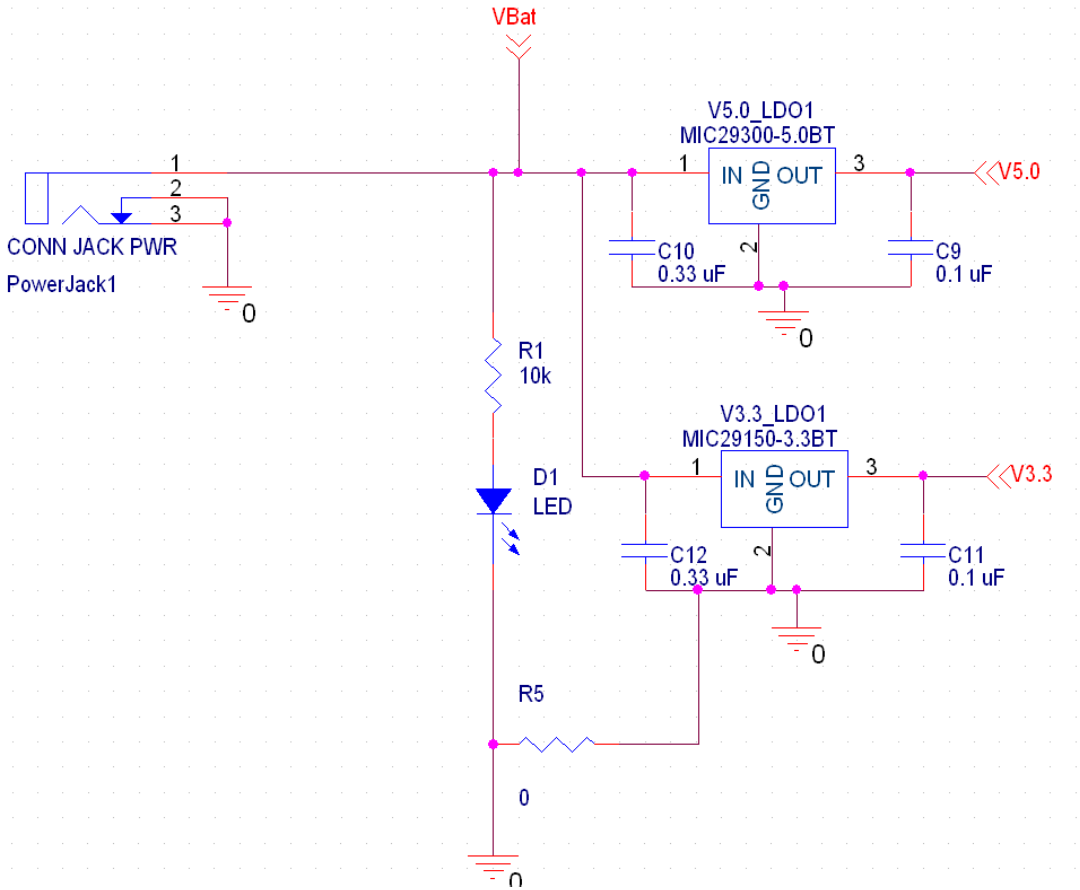


Figure C-1 Power Schematic

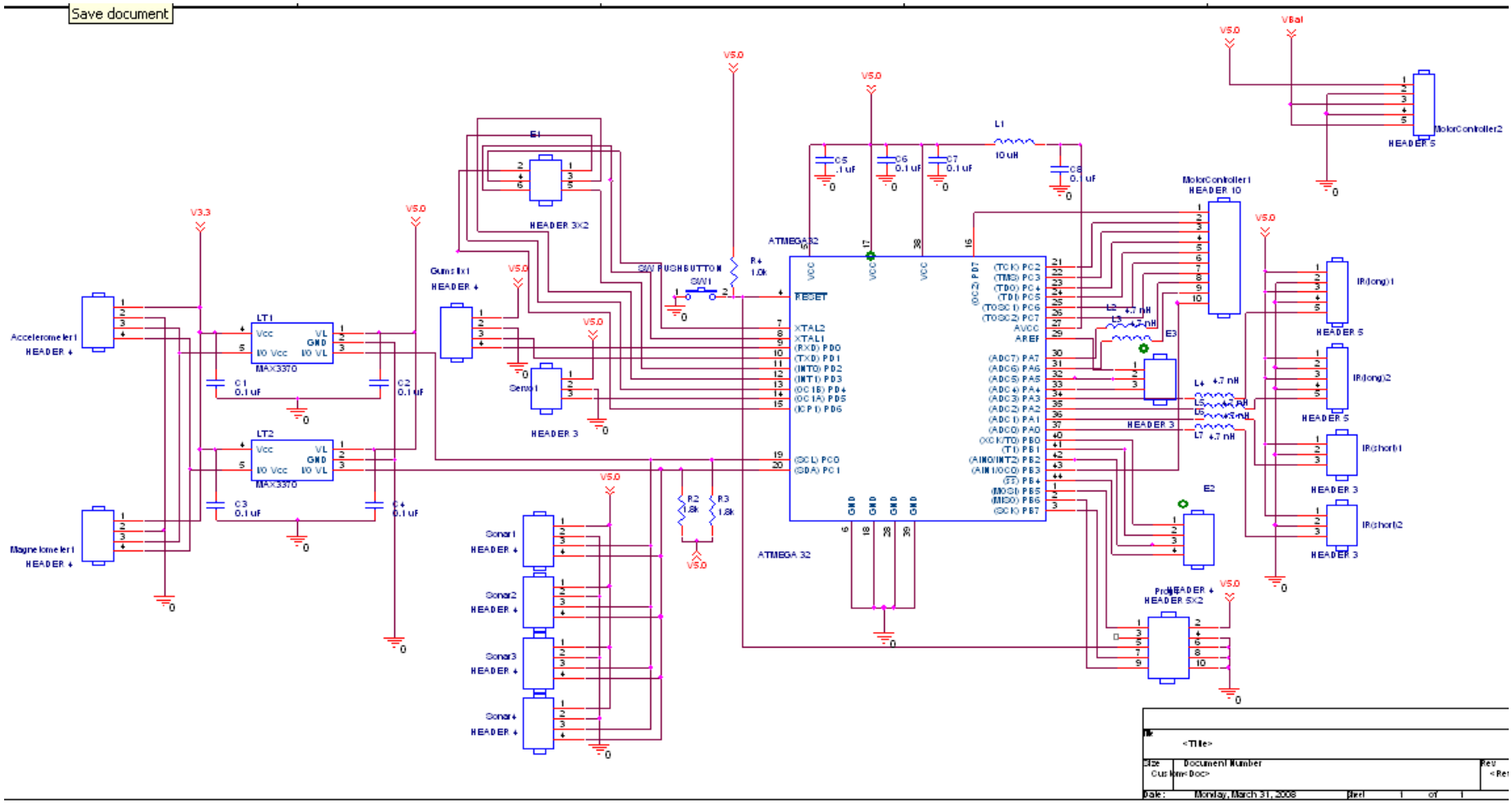


Figure C-2 ATmega32 Schematic

Appendix D: PCB Layout Top and Bottom Copper

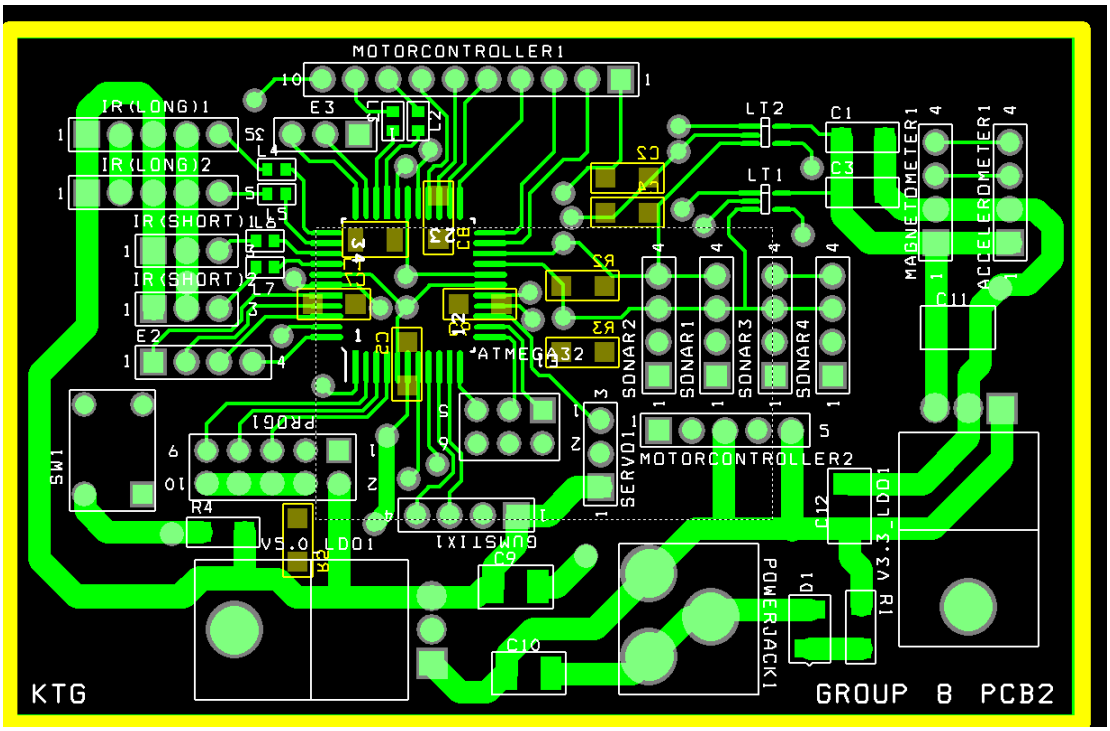


Figure D-1 Top of PCB

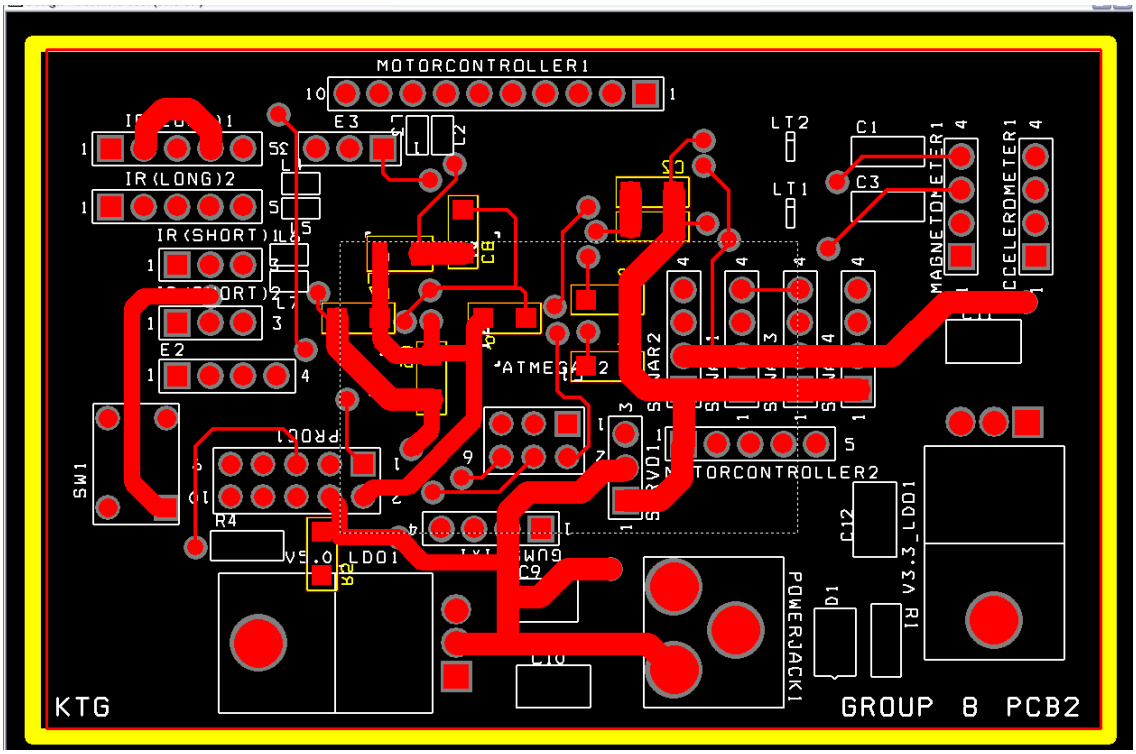


Figure D-2 Bottom of PCB

Appendix E: Parts List Spreadsheet

Table E-1 – Parts List Spreadsheet

Vendor	*Manufacturer*	*Part No.*	*Description*	*Unit Cost*	**Qty**	*Total Cost*
Mouser	Atmel	Atmega32	MicroController	6.40	1	6.40
Acroname	Devantech	SRF02 Sensor	Sonar with ok resolution	26.50	5	106.00
Acroname	Sharp	R302-GP2Y0A700K0F	Long range IR	19.50	2	39.00
Acroname	Sharp	R144-GP2Y0A02YK	Short range IR	12.50	2	25.00
Sparkfun	ST	LIS3LV02DQ	Accelerometer	43.95	1	43.95
Sparkfun	HoneyWell	HMC6352	Magnetometer	59.95	1	59.95
Gumstix	Gumstix	XL6P	Embedded Computer	269.00	1	269.00
Acroname	Acroname	R298-1T-SERVO	Servo with 360 degrees turn	18.00	1	18.00
Gumstix	Gumstix	Verdex XL6P	Embedded Computer	169.00	1	169.00
Gumstix	Gumstix	netwifimicroSD FCC	Expansion Board – wifi	99.00	1	99.00
Gumstix	Gumstix	Breakout-vx	Expansion Board	27.50	1	27.50
Future Electronics	Micrel Inc	MIC29150-3.3WT	LDO Regulator 3.3V	3.42	1	
Health Stores			Magnetic Shielding Foil 15in wide, 0.010 in thick	28.95	2	57.90
Pololu	Pololu	MD03A # 708	Dual VNH2SP30 motor driver carrier	59.95	1	59.95
Digikey	Micrel Inc	MIC29300-5.0BT-ND	LDO Regulator 5.0V @ 3A	4.25	1	4.25
Digikey	Linear Technology	LT1083CP-5#PBF-ND	LDO Regulator 5.0V @ 7.5A	13.50	1	13.50
Digikey	AVX Corporation	TAJA104K035R	Cap Tantalum 0.10uF	0.87	2	1.74
Digikey	Vishay/Sprague	293D334X9035A2TE3	Cap Tantalum 0.33uF	0.10	2	0.20
Newark	Maxim	Max3370EXK+T	Level translators	2.10	2	4.20
Digikey	Lite-On Inc	SMD-LTST-C171TBKT	LED	0.62	1	0.62
Mouser	Kemet	C1206F104K3RACTU	Ceramic Capacitor- 1206 - 0.10uF	0.47	8	3.74
Digikey	EPCOS Inc	B82422H1103K	Inductor 10uH - 1210	0.56	1	0.56
Digikey	EPCOS Inc	B82496C3479J	Inductor 4.7nH 0603	0.32	6	1.92

Digikey	Assmann Elec Inc	AWHW10G-0202-T-R	Conn Header 10 Low Pro 10 pins	0.59	1	0.59
Mouser	Vishay	CRCW12061K80JNEA	Resistor 1.8K 1206	0.08	2	0.16
Mouser	Vishay	CRCW120610K0FKEA	Resistor 10K 1206	0.10	1	0.10
Mouser	Vishay	CRCW12061K00FKEA	Resistor 1.0K 1206	0.10	1	0.10
Digikey	CUI Inc	PJ-006A	Conn PWR JACK	0.45	1	0.45
Digikey	Samtec Inc	BBL-132-G-E	Conn Header .100 32 pins	6.58	3	19.74
Mouser	Apem	MJTP1230B	Pushbutton	0.16	1	0.16
Digikey	Panasonic	ECE-A1CN220U	Cap Elect 22uF	0.24	1	0.24
Digikey	Panasonic	ECE-A1CN470U	Cap Elect 47uF	0.31	1	0.31
Digikey	Panasonic	ECE-A1CN101U	Cap Elect 100uF	0.43	1	0.43
Lynxmotion	Lynxmotion	GHM-07	Gear Head Motor	16.50	4	66.00
Lynxmotion	Lynxmotion	MMT-01	Aluminum Motor Mount	7.95	2	15.90
Lynxmotion	Lynxmotion	NFT-07	Foam Tire	5.36	2	10.72
Lynxmotion	Lynxmotion	HUB-06	Mounting Hub	8.00	2	16.00
Pro Plastics	GE	Lexan 9034	Plexiglass 36”x36”x0.125”	55.00	1	55.00
Hobby City	Polyquest	2S Lipoly	2200mAh, 7.4 V battery	55.00	1	55.00
Logitech	Logitech	QuickCam Orbit AF	Camera	129.99	1	129.99
				TOTAL		**$1382.27**

Appendix F: Software Listing

Change Log

```
********************************************************************************
**********************
Date        Rev   Author     Message
================================================================================
======================
2008-04-24  147   nelson11   changed: Idle timeout message to every 10s.changed:
Cleaned up message handling, and associated logging.
2008-04-24  146   jwildey    modified: uncommented go commands from state machine.
2008-04-24  145   nelson11   changed: Commented out serial read code for
debugging.added: Try to stop omar before we close down serial service.
2008-04-24  144   jwildey    modified: deleted the '\' after the last .o file and
deleted the terminal and fuse bit modes
2008-04-24  143   nelson11   changed: Simplified uart comms with uC.
2008-04-24  142   jwildey    Added code to get a command from the gumstix.  gumstix
will send 0x55 for GO and 0xAA for STOP.  also commented out default state in state
machion.
2008-04-24  141   nelson11   changed: Cosmetic.
2008-04-23  140   nelson11   removed: Excessive debugging statements.
2008-04-23  139   nelson11   "fixed": Turns out serial callback has been working fine
all along...I'm an idiot :(removed: Tons of unnecessary debug/commented code.
2008-04-23  138   nelson11   changed: rewrote serial callback in a less stupid way.
(Still causes problems :( )
2008-04-23  137   jwildey    added terminal modes to make file as well as hfuse and
lfuse write option to 0xd9 and 0xe4 respectively
2008-04-23  136   nelson11   added: send function for use in network -&gt; serial
communication.added: lock around serial descripter (messes up network thread...)fixed:
More scary messages after signal is caught on ARM.changed: Open serial port for non-
blocking access.
2008-04-23  135   nelson11   added: support for "go" callback in network
service.changed: "die" callback doesn't need any args.
2008-04-23  134   nelson11   fixed: Got rid of scary message when we catch a signal on
ARM.changed: Made client socket a member.added: "go" command and callback to serial to
handle it (causes problems...)
2008-04-23  133   rtoepfer   modified: added the stdlib.h for the snprintf debug
statement.
2008-04-23  132   rtoepfer   modified: added sonar structure and some sonar functions
to initialize, avg, and find the minimum distance. Added code to detect when we are
running close to a wall. The middle sonar sensor seems to not go above 51cm for some
odd reason. Maybe its picking up ghost readings from the other sensors. The side
detection wasnt tested since the speed was always slow due to the middle sensor not
functioning properly. Needs further debug/testing of middle sensor and side sensors.
2008-04-23  131   rtoepfer   Modified: added terminal mode definition.
2008-04-23  130   nelson11   added: packet structures for network and serial as well
as some enumerated types to define packets and commands.
2008-04-23  129   rtoepfer   modified: took out the accel, mag, and sonar code and put
in respective .c/.h files.
2008-04-23  128   rtoepfer   added: broke up the i2c.c and i2c.h files to smaller more
manageable files. seperated the magnetometer, accelerometer, and sonar i2c code into
seperate files.
2008-04-23  127   nelson11   added: some basic layout components for image recognition
service.
2008-04-22  126   rtoepfer     modified: reverted back to revision 108 to demonstrate
the best object avoidance. Modified the left and right speeds to go straighter.
2008-04-22  125   rtoepfer   modified: added uart.h to print status values for debug.
2008-04-18  124   nelson11   fixed: valgrind was complaining about delete being used
instead of delete[].
2008-04-18  123   nelson11   added: More callback stuff from previous commit.
```

```
2008-04-18  122   nelson11   added: Dabbled with callbacks a bit. Sending "die" to the
network kills omar.
2008-04-18  121   nelson11   changed: Compile client with gcc instead of g++.
2008-04-18  120   nelson11   removed: Commented code, unused variables.
2008-04-16  119   nelson11   removed: More unneeded debug statements.
2008-04-16  118   nelson11   fixed: Don't compile client for ARM, that's silly.
2008-04-16  117   nelson11   removed: Unneeded debug statements.
2008-04-16  116   nelson11   fixed:  Some debug stuff was outside the DEBUG #ifdef
block.
2008-04-16  115   nelson11   fixed: arm target wasn't being deleted
2008-04-16  114   nelson11   fixed: Deadlock if bad things happen on ARM. (Apparently
select() catches signals under ARM)
2008-04-10  113   rtoepfer   changed: added code for PID loop and new throttling code.
attempt at making it use compass to drive straight. Code doesnt work properly.
2008-04-10  112   rtoepfer   updated: updated the file to contain the PID loop.
tweaked the PID loop some. still not 100%.
2008-04-10  111   rtoepfer   changed: changed get_compass to no longer pass the
address. the address is static.
2008-04-10  110   rtoepfer   changed: change get_compass to no longer need the address
passed. address is static.
2008-04-10  109   jwildey    Modified more of the PID loop, still not done yet though.
2008-04-10  108   rtoepfer   changed: dont forget to comment the print statements.
this code seems to work well, except the extreme angle.
2008-04-09  107   rtoepfer   changed: adde value to detect incorrect acceleration.
2008-04-09  106   rtoepfer   changed: changed i2c.c to not error out when it errors(i
think). main was probably changed to read different address(not important).
2008-04-09  105   rtoepfer   added: initial add of servo code. 20Hz frequency, change
OCR1A to vary duty cycle. goes from 650-2450 then reverses. Boundaries seem to be
greater than the servo extremes. LDO gets really hot. need to change the max/min
values for OCR1A to match servo boundaries.
2008-04-09  104   rtoepfer   removed: deleted files that TRENT STUPIDLY inserted.
Still trying to teach trent unix... he will get there.
2008-04-09  103   rtoepfer   added: initial add of Low range IR code. Reads values
from the adc. Currently using the 2.56V internal AREF. Trent says it works but the
values dont match the ones in teh manual.
2008-04-08  102   rtoepfer   changed: added states for left and right. seems to work
fairly well. Corner case where it approaches a wall at a small angle still hits the
wall.
2008-04-08  101   rtoepfer   fixed: SILENT NOT SLIENT...TRENT...ARE U ALIVE...
RETARDED???
2008-04-08  100   nelson11   added: "silent" install target to stop the motors while
we debug.
2008-04-08  99    rtoepfer   changed: 3 sonar sensors on the front. scans 3 sensors
and detects objects. speed throttles at different object distances. turns in right
direction to avoid walls.
2008-04-08  98    rtoepfer   changed: moved a 3 lines to the beginning of timer_init()
that were in main. they set the Data directions for PORTC,D and 1 other.
2008-04-08  97    rtoepfer   changed: changed makefile to define F_CPU=8000000UL. used
for delay.h.
2008-04-07  96    rtoepfer   Modified: initial test of going until sonar senses wall,
then turn, then go. It currently senses walls, but doesnt turn enough,          then
it gets within the sonars minimum distance and hits the wall. Need to try different
object distance thresholds.
2008-04-07  95    rtoepfer   Modified: added timer.o.
2008-04-07  94    rtoepfer   Modified: copied timer.h from Motorcontrol repo.
originally was only for testing the servo. Now included for motorcontrol.
2008-04-07  93    rtoepfer   Modified: added support functions for UART RX. added
signal handler, uint8_t get_char() and uint8_t get_str(buff, size).
2008-04-07  92    rtoepfer   added: initial add of code for controlling motors.
2008-04-07  91    nelson11   added: changes to reflect inclusion of client.
2008-04-07  90    nelson11   removed: Binary file from last commit.
2008-04-07  89    nelson11   added: Test client for network service. VERY simple.
2008-04-07  88    nelson11   changed: Inactivity nag message to every 5s from 1s.
```

```
2008-04-07  87    nelson11   fixed: Forgot to uncomment the framerate limiting code.
2008-04-07  86    nelson11   fixed: Network actually works now. I'd rather not say why
it was broken...
2008-04-07  85    nelson11   added: Signal handling for SIGINT (ctrl+c)
2008-04-05  84    nelson11   fixed: Used int instead of uint8_t.
2008-04-05  83    nelson11   fixed: missed a #define
2008-04-05  82    nelson11   updated: More new makefiles!
2008-04-05  81    nelson11   fixed: Compile error (more extern misuse);
2008-04-05  80    nelson11   fixed: Compile error (misused extern).
2008-04-05  79    nelson11   added:  Cool new makefile for this project too
2008-04-05  78    jwildey    Added i2c readings to main.c to figure out that the
magnetometer is dead.  modified the pid loop a little bit to accept 2 vars and to
return 16 bit value.
2008-04-05  77    nelson11   fixed: Was trying to writete .elf to the uC not the
.srec.
2008-04-05  76    nelson11   added: New more useful makefiles
2008-04-05  75    nelson11   updated: ATMega32 related makefiles to use new avrisp2.
NOTE: This breaks install on the STK500.  I'll fix it properly tomorrow.
2008-04-05  74    jwildey    Have added timer.c, uart.h, uart.c, i2c.c, i2c.h,
MControl.c, MControl.h.  Added these to start creating PD loop for motor controller.
Will use magnetometer to get heading and PD loop will update accordingly eventually.
Got timer working correctly.  Timer will be used for timestamp for derivative portion
of controller.
2008-04-04  73    nelson11   added: Changelog generation script.
2008-04-04  72    nelson11   Changed: Wait forever (just for testing)
2008-04-04  71    nelson11   Added: Network class.
2008-04-04  70    nelson11   Added: Basic support for network service.
2008-04-04  69    rtoepfer   modified: attempted to change the code so that the get
data command could be called and later read the data. Works for all devices
except the sonar for some unkown reason. Sonar will not communicate in this fashion.
Delays were added but did not change          anything.
2008-04-04  68    rtoepfer   added: timer.h added to test the noise coming from the
servo.
2008-04-04  67    nelson11   added: Basic definition of network class.
2008-04-04  66    nelson11   fixed: Allocate buffers on initialization instead of
statically. Saves stack space.
2008-04-04  65    nelson11   fixed: typo and missing include.
2008-04-04  64    nelson11   changed: arm target binary is now named omar-arm
2008-04-03  63    rtoepfer   modified: changed main to read the IR sensors and the
makefile to use the sstk500 board for programing. change to v2 for the avrisp.
2008-04-03  62    rtoepfer   added: initial add of adc code to main loop.
2008-04-02  61    nelson11   fixed: build / run commands should work in kdevelop now
2008-04-02  60    nelson11   changed:  Mostly cosmetic
2008-04-02  59    nelson11   fixed:  NULL member pointers in constructor.
2008-04-01  58    rtoepfer   added: initial check in of main Sensor board code.
Currently reads Magnetometer, accelerometer, and Sonar code and outputs to uart.
2008-04-01  57    rtoepfer   added: initial import of working magnetometer code.
outputs full 360 deg heading to uart.
2008-04-01  56    rtoepfer   added: initial check in of working accelerometer code.
reads X,Y,Z forces.
2008-04-01  55    rtoepfer   updated: i2c test code working, reading lower byte of
data only.
2008-04-01  54    rtoepfer   removed: unecessary backup files.
2008-03-31  53    nelson11   added: Kdevelop project file.
2008-03-31  52    nelson11   fixed: "distclean" target actually cleans up everything
now
2008-03-31  51    nelson11   changed: Camera class to reflect changes in v4l.
2008-03-31  50    nelson11   changed:  Cleaned up makefiles a bit.  They're a little
more flexible now.
2008-03-31  49    nelson11   changed: Renamed v4l.cpp back to v4l.c.
2008-03-31  48    nelson11   fixed: Compile error if DEBUG not defined.
2008-03-31  47    nelson11   changed:  Run for 5s instead of 3.
```

```
2008-03-31  46    nelson11   changed: Makefiles now detect arch change and run make
clean only as necessary.added: Makefiles now auto finds dependencies. Just add object
to OBJS variable in Makefile.
2008-03-30  45    nelson11   fixed: Linking problem with libjpeg.
2008-03-30  44    nelson11   added: target to make for ARM.
2008-03-28  43    nelson11   removed:  Commented test code that uses libjpeg so we can
link
2008-03-28  42    nelson11   fixed: Compile error.
2008-03-28  41    nelson11   changed: make for x86 again cause I can't get libjpeg to
link in for some dumb reason.
2008-03-28  40    nelson11   changed: make to use arm compiler. WON'T WORK ON X86!
2008-03-28  39    nelson11   added:  Camera service actually takes pictures now!
added:  Test function to store images as jpeg.
2008-03-28  38    nelson11   fixed:  Check for successful oject creation didn't work
at all.
2008-03-27  37    nelson11   changed: Makefile is a little more flexible for
subdirectories now.
2008-03-27  36    nelson11   modified: I'm just using this to test stuff ATM.  Changed
it to start the "omar" control thread which then starts the other services.
2008-03-27  35    nelson11   added:  Serial and camera services to the "omar" control
thread.
2008-03-27  34    nelson11   removed:  All of the debug code this time...fixed:
Duplicate log message.
2008-03-27  33    nelson11   removed: Duplicate log message.fixed: Potential deadlock
if select() fails miserably.
2008-03-27  32    nelson11   added:  Missd a header in the last commit.removed:  Some
debug code that forced camera service to succeed without camera attached.
2008-03-27  31    nelson11   added:  Most of the code for the camera service class
(needs testing).
2008-03-27  30    nelson11   fixed:  Makefile didn't relink when most files were
modified.
2008-03-27  29    nelson11   changed:  Moved v4l.c to v4l.cpp so it will be compiled
with g++ and actually link with the rest of the code.updated:  Makefile to reflect the
above.
2008-03-27  28    nelson11   removed:  Unnecessary getStatus() function.fixed:  Call
close_serial() if we fail to create thread.changed:  serial_initialize() doesn't need
the device passed to it, device is a member.
2008-03-27  27    nelson11   changed: Cosmetic.
2008-03-27  26    nelson11   updated: Makefile to reflect v4l and network class
additions
2008-03-27  25    nelson11   added: v4l files for camera. Ganked from aerial robotics
repo, see ../../refs.
2008-03-27  24    nelson11   added: network class template files.
2008-03-27  23    nelson11   added: old od (camera) lib from aerial robotics repo.
2008-03-24  22    nelson11   added: Overloaded ! operator to allow for checking
success of serial object creation.
2008-03-23  21    nelson11   updated: Makefile to handle logger.fixed: compile error
2008-03-23  20    nelson11   added: Basic support for serial service class.
2008-03-23  19    nelson11   added: Initial commit of logger class.
2008-03-10  18    nelson11   added:  messages.h. define interface with uC here.
2008-03-07  17    nelson11   added: Initial code base for the gumstix.
2008-03-07  16    nelson11   removed:  More binaries...You guys new or something?
2008-03-07  15    nelson11   removed: Binaries shouldn't be under revision control
either...
2008-03-07  14    nelson11   removed: vim swap file...BAD JOSH!!
2008-03-06  13    jwildey    Added capability for all motors and both H-Brdiges.
Added functions to go forward, reverse, left, right, and brake.  Will restructure so
that prototype definitions are in header file. Also will add capability for 2nd PWM
and push button start to help see how straight OMAR doesn't go.
2008-03-03  12    jwildey    Created test code to start driving the Motors.  Set up
PWM on the OC0 pin on the ATmega32 running at 3.9ish kHz. PC6 and PC7 to be GP outs to
control H-bridges.  Only code to drive one H-bridge.
2008-02-13  11    rtoepfer   removed: No clue how this got checked in here...
```

```
2008-02-13  10    rtoepfer   moved: GP2Y0A700K to trunk too
2008-02-13  9     rtoepfer   moved: srf02 into trunk where it belongs
2008-02-13  8     rtoepfer   added: Initial commit of SRF02 test code.
2008-02-13  7     rtoepfer   Trent is going to kill me
2008-02-13  6     rtoepfer   I dicked up the repo :)
2008-02-13  5     rtoepfer   added: Initial commit of SRF02 Sonar test code.
2008-02-11  4     rtoepfer   added: Long range sensor IR (GP2Y0A700K). wrote initial
driver to initialize adc and read adc.        a test program (main.c) was written to
test the adc with the IR sensor.
2008-02-08  3     rtoepfer   removed: initially put into wrong directory. New
location: Sensors.
2008-02-08  2     rtoepfer   added: initial import of sample i2c code for srf02.
2008-02-02  1     nelson11   Initial repository creation
```

Actual Code

```
###### Subvehicle/Sensors/trunk/GP2Y0A700K/adc.h #####

#ifndef __MY_ADC__
#define __MY_ADC__

#include <avr/interrupt.h>
#include <avr/io.h>

extern uint8_t uval, lval;

void adc_init();

#endif //__MY_ADC__

###### Subvehicle/Sensors/trunk/GP2Y0A700K/main.c #####

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


#include "uart.h"
#include "adc.h"

int main()
{

  uint16_t i, a[10], size = 0, p = 0, sum;
  char c[10];
  memset((void*)c,0,10);
  memset((void*)a, 0, 10*sizeof(uint16_t));
//    WDTCR |= (1 << WDTOE) | (1 << WDE);
//    WDTCR &= (0<< WDE);
  /* Initialize UART peripherial, MUST BE CALLED */
  cli();
  uart_init();
  adc_init();
  sei();
  put_char('A');
  uval = 0;
  lval = 0;

 while(1){
 /* Delay so we don't send stuff too fast*/
#if 1
 for(i = 1; i != 0; i++)
    {
      asm("nop");
      asm("nop");
      asm("nop");
      asm("nop");
      asm("nop");
      asm("nop");
      asm("nop");
      asm("nop");
      asm("nop");
      asm("nop");
      asm("nop");
```

```
    }
#endif
//    ADCSRA |= (1 << ADSC);

    put_str("Upperval = ",11);
    a[p] = read_adc();
    p = (p + 1) % 10;
    if (size < 10) size++;

    sum = 0;
    for (i= 0; i < size; i++) sum += a[i];

    snprintf(c, 10,"%d\n\r",(int)(sum/size));
   // itoa((int)uval, c, 10);
    put_str(c,10);
  }
}
```

Subvehicle/Sensors/trunk/GP2Y0A700K/test.c

```
#include <stdlib.h>
#include <stdio.h>

int main ( ) {
  int i = 45;
  char s[10];

  itoa(i, s, 10);

  printf("%s\n", s);

  return 0 ;
  }
```

Subvehicle/Sensors/trunk/GP2Y0A700K/uart.c

```
#include <avr/interrupt.h>
#include <avr/io.h>
#define BAUD 12
#include "uart.h"

/* UART variables */
static volatile uint8_t tx_read;
static volatile uint8_t tx_write;
static volatile uint8_t tx_buff[TX_BUFF_SIZE];

void uart_init()
{
  /* Enable the UART for sending and receiving */
  sbi(UCSRB, RXEN);
  sbi(UCSRB, TXEN);

  tx_read = tx_write = 0;

  /* Set baud rate registers */
  UBRRH = (((uint16_t)BAUD) & 0xff00) >> 8;
  UBRRL = ((uint16_t)BAUD) & 0xff;

  /* Enable Rx interrupt */
  sbi(UCSRB, RXCIE);
}

/* UART tx interrupt */
```

```
SIGNAL(SIG_UART_DATA)
{
  /* Tx buffer is empty */
  if(tx_read == tx_write)
  {
    cbi(UCSRB, UDRIE);
    return;
  }

  UDR = tx_buff[tx_read];

  tx_read = tx_read + 1 >= TX_BUFF_SIZE ? 0 : tx_read + 1;
}

void put_char(uint8_t c)
{
  uint8_t tmp_write;

  tmp_write = tx_write + 1 >= TX_BUFF_SIZE ? 0 : tx_write + 1;

  /* Buffer is full */
  if(tmp_write == tx_read)
  {
    sbi(UCSRB, UDRIE);
    return;
  }


  tx_buff[tx_write] = c;
  tx_write = tmp_write;

  /* Enable Tx interrupt to start sending */
  sbi(UCSRB, UDRIE);

  return;
}

void put_short(uint16_t s)
{
  put_char(((uint8_t)s & 0xFF));
  put_char((uint8_t)(s >> 8));
}

void put_str(char *str, uint32_t len)
{
  uint32_t i;

  for(i = 0; i < len; i++)
    put_char(str[i]);
}


###### Subvehicle/Sensors/trunk/GP2Y0A700K/adc.c #####

#ifndef ADC__H__
#define ADC__H__

#include <avr/interrupt.h>
#include <avr/io.h>
#include "adc.h"

uint8_t uval,lval;
```

```
void adc_init(void)
{
  ADCSRA = 0x00;
  PORTA = 0x00; DDRA = 0xFE; //port a is input others are output
  ADMUX = 0xC0; //port0 enabled
  ADCSRA = 0x83; //enabled.start.auto trigger.interrupt flag. interupt
enabled.prescalars
 // SFIOR = 0x00; //free running mode, set auto trigger to interupt flag
 // ADCSRA |= (1 << ADEN);
 // ADCSRA |= (1 << ADSC);
 // return;
}

void adc_isr(void)
{
  lval = (uint8_t)ADCL;
  uval = (uint8_t)ADCH;
  return;
}

uint16_t read_adc(void)
{
  uint16_t i;
  uint8_t x;
  ADCSRA |= (1 << ADSC);
  while( (ADCSRA & ADSC) >> ADSC);

  x = ADCL;
  i = 0x0003 & ADCH;
  i <<= 8;
  i = i | (0x00FF & x);
  return i;
}

#endif // ADC__H__

###### Subvehicle/Sensors/trunk/GP2Y0A700K/uart.h #####

#ifndef __MY_UART__
#define __MY_UART__

#include <avr/interrupt.h>
#include <avr/io.h>

#define TX_BUFF_SIZE 256

#ifndef BAUD
/* If different baud rate is desired do #define BAUD ??
   BEFORE #include "uart.h" in your source */
#define BAUD 51 /* 9600 baud @ 8MHz normal speed */
#endif

#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif

/* Initialize UART, MUST BE CALLED */
void uart_init();
/* Write 8-bit value to UART */
```

```
void put_char(uint8_t c);
/* Write 16-bit value to UART */
void put_short(uint16_t s);
/* Write string of length len to UART */
void put_str(char *str, uint32_t len);

#endif /* __MY_UART__ */

###### Subvehicle/Sensors/trunk/LIS3L/i2c.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
#include "i2c.h"
#include "uart.h"


void i2c_init()
{
  PORTC = 0x03;  //port0 and port1 internally pulled high
  //TWSR = (0 << TWPS0) & (0 << TWPS1); //prescalar 1
  TWBR = 8;//(F_CPU / 100000UL - 16) / 2; //0x02;
}//end init



int i2c_transmit(uint8_t addr, uint8_t reg, uint8_t data)
{
  //int status = 0;

  if ( send_start() == ERROR)
    return ERROR;
  if ( send_addr_t(addr) == ERROR)
    return ERROR;
  if ( send_data(reg) == ERROR)
    return ERROR;
  if (send_data(data) == ERROR)
    return ERROR;

  send_stop();


  return OK;
}//end i2c_transmit(addr, data);

int8_t i2c_read(uint8_t addr, uint8_t reg)
{
  uint8_t data = 0;

  if (send_start() ==ERROR)
    return ERROR;
  if (send_addr_t(addr) == ERROR)
    return ERROR;
  if (send_data(reg) == ERROR)
    return ERROR;
  if (send_repeat() == ERROR)
    return ERROR;
  if (send_addr_r(addr) == ERROR)
    return ERROR;
  if (get_nack_data() == ERROR)
    return ERROR;

  data = TWDR;
  send_stop();
```

```
  return data;
}//end i2c_read(addr, reg);


uint8_t i2c_read_accel(uint8_t addr, uint8_t reg, int16_t *data)
{
  if (send_start() == ERROR)
    return ERROR;
  if (send_addr_t(addr) == ERROR)
    return ERROR;
  if (send_data(reg) == ERROR)
    return ERROR;
  if (send_repeat() == ERROR)
    return ERROR;
  if (send_addr_r(addr) == ERROR)
    return ERROR;

  if (get_ack_data() == ERROR)
    return ERROR;
  data[0] = TWDR;
  if (get_ack_data() == ERROR)
    return ERROR;
  data[0] += TWDR << 8;

  if (get_ack_data() == ERROR)
    return ERROR;
  data[1] = TWDR;
  if (get_ack_data() == ERROR)
    return ERROR;
  data[1] += TWDR << 8;

  if (get_ack_data() == ERROR)
    return ERROR;
  data[2] = TWDR;
  if (get_nack_data() == ERROR)
    return ERROR;
  data[2] += TWDR << 8;

  send_stop();
  return OK;

}//end i2c_read_accel

/******* send_repeat(); ************************/
int send_repeat()
{
  TWCR = 0 | (1 << TWSTA) | (1 << TWINT) | (1 << TWEN); //repeat start code

  // wait for repeat start sent
  while ( !(TWCR & (1 << TWINT)));

  // check repeat start sent correct
  if ((TWSR & 0xF8) != TW_REP_START)
    return ERROR; //put_str("REP_START_ERROR\n\r");

  return OK;
}//end send_repeat()



/******* send_start() ******************************/
int send_start()
{
```

```
  /* send start */
  TWCR = 0 | (1 << TWINT) | (1 << TWSTA) | (1 << TWEN); //send start condition

  /* wait for start sent */
  while (! (TWCR & (1 << TWINT))); // wait for TWINT flag set.(indicates start sent.

  /* check start sent correct */
  if ((TWSR & 0xF8) != START)
    return ERROR; //put_str("START_ERROR\n\r");

  return OK;
}//end send_start



/****** send_addr_t(uint8_t addr); *******************/
int  send_addr_t(uint8_t addr)
{
  char s[25];

  /* send addr */
  TWDR = addr;
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT to start transmission of
addresss.

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT)))
  {
//    put_str("send_addr_t while loop\n\r");// wait for TWINT set. indicates Slave
addr sent and ACK received.
  }

  /*check status = slave sent/ack received */
  if ((TWSR & 0xF8) != TW_MT_SLA_ACK)
    return ERROR;

  return OK;
}//end send_addr


/****** send_addr_r(uint8_t addr); *******************/
int send_addr_r(uint8_t addr)
{
  /* send addr */
  TWDR = addr | 0x01;

  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWEA); //clear TWINT to start
transmission of addresss. send ACK

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT set. indicates Slave addr sent
and ACK received.

  /*check status = slave sent/ack received */
  if ((TWSR & 0xF8) != TW_MR_SLA_ACK)
    return ERROR;  //put_str("ADDR_SENT_ERROR\n\r");

  return OK;
}//end send_addr


/*************************************
 *      int send_data(uint8_t data);
```

```
 ************************************/
int send_data(uint8_t data)
{
  /* send data */
  TWDR = data;
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT bit to start data transmission.

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, ACK
received.

  /* check status = MT_DATA_ACK */
  if ( (TWSR & 0xF8) != TW_MT_DATA_ACK)
    return ERROR; // put_str("DATA_SENT_ERROR\n\r");

  return OK;
}// end send_data



/********** send_stop(); *****************/
void send_stop()
{
  /* send STOP condition */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);

  //return OK;
}



/********* int get_data(); ***************/
int get_nack_data()
{
  /* get data */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT bit to start data read. send
NACK.

  /* wait for ACK sent */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, NACK sent.

  /* check status = MR_DATA_NACK */
  if ( (TWSR & 0xF8) != TW_MR_DATA_NACK)
    return ERROR; //put_str("DATA_REC_ERROR\n\r");

  return OK;
}

/********* int get_data(); ***************/
int get_ack_data()
{
  /* get data */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWEA); //clear TWINT to start
transmission of addresss. send ACK

  /* wait for ACK sent */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, NACK sent.

  /* check status = MR_DATA_NACK */
  if ( (TWSR & 0xF8) != TW_MR_DATA_ACK)
    return ERROR; //put_str("DATA_REC_ERROR\n\r");

  return OK;
```

```
}



###### Subvehicle/Sensors/trunk/LIS3L/main.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
#define F_CPU 800000UL
#include "i2c.h"
#include "uart.h"

int main()
{
  uart_init();
  while (1) {
    put_str("Mary had a little lamb\r\n");
  }
  return 0;
}
#if 0
  int16_t j,i=0, size=0;
  //uint8_t data = 0x00;
  int16_t data[3] = {0,0,0};
  int32_t avgx[11] = {0}, avgy[11] = {0}, avgz[11] = {0};
  int16_t totalx, totaly, totalz = 0;

  char s[32];

  /* init uart and i2c */
  uart_init();
  i2c_init();
  sei();

  put_str("HI my name is RJ from main\n\n\r");

  i2c_transmit(0x3A, 0x20, 0xB7);

  while(1)
  {
    _delay_ms(2500);

#if 0
    if ( i2c_read_accel(0x3A, 0x88, data) == ERROR)
    {
      put_str("failed\n\r");
      send_stop();
    }
    else
    {
      snprintf(s, 32, "X=%4d Y=%4d Z=%4d\r", data[0], data[1], data[2]);
      put_str(s);
     // snprintf(s, 25, "Y=%d\n\r", data[1]);
     // put_str(s);
     // snprintf(s, 25, "Z=%d\n\r", data[2]);
     // put_str(s);
    }
  }
#else
    if ( (data[0] = i2c_read(0x3A, 0x28)) == ERROR)
    {
      put_str("failed\n\r");
```

```
      send_stop();
    }
    if ( (data[0] += i2c_read(0x3A, 0x29) << 8) == ERROR)
    {
      put_str("failed\n\r");
      send_stop();
    }

    if ( (data[1] = i2c_read(0x3A, 0x2A)) == ERROR)
    {
      put_str("failed\n\r");
      send_stop();
    }
    if ( (data[1] += i2c_read(0x3A, 0x2B) << 8) == ERROR)
    {
      put_str("failed\n\r");
      send_stop();
    }

    if ( (data[2] = i2c_read(0x3A, 0x2C)) == ERROR)
    {
      put_str("failed\n\r");
      send_stop();
    }
    if ( (data[2] += i2c_read(0x3A, 0x2D) << 8) == ERROR)
    {
      put_str("failed\n\r");
      send_stop();
    }

  //  snprintf(s, 32, "X=%6d Y=%6d Z=%6d \r", data[0], data[1], -(data[2]));
    put_str(s);
   #if 1
    avgx[i] = data[0];
    avgy[i] = data[1];
    avgz[i] = data[2];

    i = (i+1) % 10;

    totalx = totaly = totalz =0;

    if (size < 10) size++;

    for (j = 0; j < size; j++) {
      totalx += avgx[j];
      totaly += avgy[j];
      totalz += avgz[j];
    }
    #endif


    snprintf(s, 32, "X=%4d Y=%4d Z=%4d \r", totalx/size, totaly/size, -(totalz)/size);
    put_str(s);
  }
#endif
 return(0);
}//end main
#endif


###### Subvehicle/Sensors/trunk/LIS3L/uart.c #####

#include <avr/interrupt.h>
```

```
#include <avr/io.h>
#define BAUD 12
#include "uart.h"

#define NULL (void*)0

/* UART variables */
static volatile uint8_t tx_read;
static volatile uint8_t tx_write;
static volatile uint8_t tx_buff[TX_BUFF_SIZE];

void uart_init()
{
  /* Enable the UART for sending and receiving */
  sbi(UCSRB, RXEN);
  sbi(UCSRB, TXEN);

  tx_read = tx_write = 0;

  /* Set baud rate registers */
  UBRRH = (((uint16_t)BAUD) & 0xff00) >> 8;
  UBRRL = ((uint16_t)BAUD) & 0xff;

  /* Enable Rx interrupt */
  sbi(UCSRB, RXCIE);
}

/* UART tx interrupt */
SIGNAL(SIG_UART_DATA)
{
  /* Tx buffer is empty */
  if(tx_read == tx_write)
  {
    cbi(UCSRB, UDRIE);
    return;
  }

  UDR = tx_buff[tx_read];

  tx_read = tx_read + 1 >= TX_BUFF_SIZE ? 0 : tx_read + 1;
}

void put_char(uint8_t c)
{
  uint8_t tmp_write;

  tmp_write = tx_write + 1 >= TX_BUFF_SIZE ? 0 : tx_write + 1;

  /* Buffer is full */
  if(tmp_write == tx_read)
  {
    sbi(UCSRB, UDRIE);
    return;
  }


  tx_buff[tx_write] = c;
  tx_write = tmp_write;

  /* Enable Tx interrupt to start sending */
  sbi(UCSRB, UDRIE);

  return;
```

```
}

void put_short(uint16_t s)
{
  put_char(((uint8_t)s & 0xFF));
  put_char((uint8_t)(s >> 8));
}

void put_str(char *str)
{
  uint32_t i = 0;

  while (str[i] != (char)NULL)
  {
    put_char(str[i]);
    i++;
  }
}


###### Subvehicle/Sensors/trunk/LIS3L/uart.h #####

#ifndef __MY_UART__
#define __MY_UART__

#include <avr/interrupt.h>
#include <avr/io.h>

#define TX_BUFF_SIZE 256

#ifndef BAUD
/* If different baud rate is desired do #define BAUD ??
   BEFORE #include "uart.h" in your source */
#define BAUD 51 /* 9600 baud @ 8MHz normal speed */
#endif

#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif

/* Initialize UART, MUST BE CALLED */
void uart_init();
/* Write 8-bit value to UART */
void put_char(uint8_t c);
/* Write 16-bit value to UART */
void put_short(uint16_t s);
/* Write string of length len to UART */
void put_str(char *str);

#endif /* __MY_UART__ */

###### Subvehicle/Sensors/trunk/LIS3L/i2c.h #####

#ifndef I2C__H__
#define I2C__H__

#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>
```

```
#include <util/twi.h>
#define F_CPU 800000UL
#include <util/delay.h>

#define START_ERROR 1
#define ADDR_SENT_ERROR 2
#define DATA_SENT_ERROR 3
#define ADDR_REC_ERROR 4
#define DATA_REC_ERROR 5
#define DATA_NACK_REC_ERROR 6
#define SENT_OK 7
#define OK    0
#define ERROR 1
#define START 0x08

void i2c_init();
int i2c_transmit(uint8_t addr, uint8_t reg, uint8_t data);
int8_t i2c_read(uint8_t addr, uint8_t reg);
uint8_t i2c_read_accel(uint8_t addr, uint8_t reg, int16_t *data);
int send_repeat();
int send_start();
void send_stop();
int send_addr_t(uint8_t addr);
int send_addr_r(uint8_t addr);
int send_data(uint8_t data);
int get_ack_data();
int get_nack_data();

#endif //I2C__H__

###### Subvehicle/Sensors/trunk/srf02/i2c.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
#include "i2c.h"
#include "uart.h"

uint8_t i2c_error;

void i2c_init()
{
  i2c_error = OK;
  //PORTC = 0x03;  //port0 and port1 internally pulled high
  //TWSR = (0 << TWPS0) & (0 << TWPS1); //prescalar 1
  TWBR = 4;//(F_CPU / 100000UL - 16) / 2; //0x02; //2 with prescalar 1 should = 100kHz
}//end init


int srf_change_addr(uint8_t curr_addr, uint8_t new_addr)
{ i2c_error = OK;
  i2c_transmit(curr_addr,0xA0);
  _delay_ms(32);
  i2c_transmit(curr_addr,0XAA);
  _delay_ms(32);
  i2c_transmit(curr_addr,0xA5);
  _delay_ms(32);
  i2c_transmit(curr_addr,new_addr);
  _delay_ms(32);

  return OK;
}

uint8_t srf_read(uint8_t addr)
```

```
{
  i2c_error = OK;
  i2c_transmit(addr, 0x51);
   _delay_ms(110);
  //range = i2c_read(addr,2) << 8;
  //_delay_ms(32);
  //_delay_ms(32);
  //_delay_ms(6);
  return(i2c_read(addr,3));


 /*
  if (send_start() == ERROR)
    i2c_error = ERROR;
  if (send_addr_t(addr) == ERROR)
    i2c_error = ERROR;
  if (send_data(0x02) == ERROR)
    i2c_error = ERROR;
  if (send_repeat() == ERROR)
    i2c_error = ERROR;
  if (send_addr_r(addr) == ERROR)
    i2c_error = ERROR;
  if (get_data() == ERROR)
    i2c_error = ERROR;

  range = TWDR << 8;
  put_str("check\n\r");
  if (send_repeat() == ERROR)
    i2c_error = ERROR;
  if (send_addr_t(addr) == ERROR)
    i2c_error = ERROR;
  if (send_data(0x03) == ERROR)
    i2c_error = ERROR;
  if (send_repeat() == ERROR)
    i2c_error = ERROR;
  if (send_addr_r(addr) == ERROR)
    i2c_error = ERROR;
  if (get_data() == ERROR)
    i2c_error = ERROR;

  range += TWDR;

  send_stop();
 */
  //return range;
}

int i2c_transmit(uint8_t addr, uint8_t data)
{ i2c_error = OK;
  //int status = 0;

  if ( send_start() == ERROR)
    i2c_error = ERROR;
  if ( send_addr_t(addr) == ERROR)
    i2c_error = ERROR;
  if ( send_data(0) == ERROR)
    i2c_error = ERROR;
  if (send_data(data) == ERROR)
    i2c_error = ERROR;

  send_stop();
```

```
  return OK;
}//end i2c_transmit(addr, data);

uint8_t i2c_read(uint8_t addr, uint8_t reg)
{ i2c_error = OK;
  uint8_t data = 0;

  if (send_start() ==ERROR)
    i2c_error = ERROR;
  if (send_addr_t(addr) == ERROR)
    i2c_error = ERROR;
  if (send_data(reg) == ERROR)
    i2c_error = ERROR;
  if (send_repeat() == ERROR)
    i2c_error = ERROR;
  if (send_addr_r(addr) == ERROR)
    i2c_error = ERROR;
  if (get_data() == ERROR)
    i2c_error = ERROR;

  data = TWDR;
  send_stop();
  return data;
}//end i2c_read(addr, reg);



/******* send_repeat(); ************************/
int send_repeat()
{ i2c_error = OK;
  TWCR = 0 | (1 << TWSTA) | (1 << TWINT) | (1 << TWEN); //repeat start code

  // wait for repeat start sent
  while ( !(TWCR & (1 << TWINT)));

  // check repeat start sent correct
  if ((TWSR & 0xF8) != TW_REP_START)
    i2c_error = ERROR; //put_str("REP_START_ERROR\n\r");

  return OK;
}//end send_repeat()



/******* send_start() ******************************/
int send_start()
{ i2c_error = OK;
  /* send start */
  TWCR = 0 | (1 << TWINT) | (1 << TWSTA) | (1 << TWEN); //send start condition

  /* wait for start sent */
  while (! (TWCR & (1 << TWINT))); // wait for TWINT flag set.(indicates start sent.

  /* check start sent correct */
  if ((TWSR & 0xF8) != START)
    i2c_error = ERROR;//put_str("START_ERROR\n\r");

  return OK;
}//end send_start



/****** send_addr_t(uint8_t addr); *********************/
```

```
int  send_addr_t(uint8_t addr)
{ i2c_error = OK;

  /* send addr */
  TWDR = addr;
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT to start transmission of
addresss.

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT set. indicates Slave addr sent
and ACK received.

  /*check status = slave sent/ack received */
  if ((TWSR & 0xF8) != TW_MT_SLA_ACK)
    i2c_error = ERROR; //put_str("ADDR_SENT_ERROR\n\r");

  return OK;
}//end send_addr


/****** send_addr_r(uint8_t addr); ********************/
int send_addr_r(uint8_t addr)
{ i2c_error = OK;
  /* send addr */
  TWDR = addr | 0x01;

  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWEA); //clear TWINT to start
transmission of addresss. send ACK

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT set. indicates Slave addr sent
and ACK received.

  /*check status = slave sent/ack received */
  if ((TWSR & 0xF8) != TW_MR_SLA_ACK)
    i2c_error = ERROR; //put_str("ADDR_SENT_ERROR\n\r");

  return OK;
}//end send_addr


/*************************************
 *      int send_data(uint8_t data);
 *************************************/
int send_data(uint8_t data)
{ i2c_error = OK;
  /* send data */
  TWDR = data;
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT bit to start data transmission.

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, ACK
received.

  /* check status = MT_DATA_ACK */
  if ( (TWSR & 0xF8) != TW_MT_DATA_ACK)
    i2c_error = ERROR; //put_str("DATA_SENT_ERROR\n\r");

  return OK;
}// end send_data
```

```
/********** send_stop(); *****************/
void send_stop()
{ i2c_error = OK;
  /* send STOP condition */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);

  //return OK;
}



/********* int get_data(); ***************/
int get_data()
{ i2c_error = OK;
  /* get data */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT bit to start data read. send
NACK.

  /* wait for ACK sent */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, NACK sent.

  /* check status = MR_DATA_NACK */
  if ( (TWSR & 0xF8) != TW_MR_DATA_NACK)
    i2c_error = ERROR; //put_str("DATA_REC_ERROR\n\r");

  return OK;
}

###### Subvehicle/Sensors/trunk/srf02/main.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
#define F_CPU 800000UL
#include "i2c.h"
#include "uart.h"

int main()
{
  uint8_t i=0;
  //int status = 0;
  //uint8_t data = 0x00;
  char s[100];
  uint8_t range = 0;
  uint8_t range1 = 0;
  uint8_t range2 = 0;
  /* init uart and i2c */
  uart_init();
  i2c_init();
  sei();

  put_str("HI my name is RJ from main\n\n\r");
  while(1)
  {
    _delay_ms(1000);
    _delay_ms(1000);
    _delay_ms(1000);
  /*
    if ( (range = srf_read(0xE0)) == ERROR)
    {
      send_stop();
    }
    else
    {
```

```
      snprintf(s, 25, "\rd=%4dcm", range);
      put_str(s);
    }
   */

   /*
    i2c_transmit(0xE4,0x51);
    if (i2c_error == ERROR)
    {
       put_str("ERROR\n\r");
       send_stop();
       continue;
    }*/
    range = srf_read(0xE0);
    _delay_ms(32);
    _delay_ms(32);
    _delay_ms(32);
    range1 = srf_read(0xE2);
    _delay_ms(32);
    _delay_ms(32);
    _delay_ms(32);
    _delay_ms(32);

    range2 = srf_read(0xE6);
    _delay_ms(32);
    _delay_ms(32);
    _delay_ms(32);
/*
    range = i2c_read(0xE4,2);
    snprintf(s, 25, "\rd=%4ucm", range);
    //put_str(s);
    if (i2c_error == ERROR)
    {
       put_str("ERROR\n\r");
       send_stop();
       continue;
    }
    else
    {
       range <<=8;
    }
    range += i2c_read(0xE4,3);
    snprintf(s, 25, "\rd=%4ucm", range);
    put_str(s);
    if (i2c_error == ERROR)
    {

       put_str("ERROR\n\r");
       send_stop();
       continue;
    }
  */
    snprintf(s, 100, "\rsonar1=%3d sonar2=%3d sonar3=%3d", range, range1, range2);
    put_str(s);

  }

 return(0);
}//end main


###### Subvehicle/Sensors/trunk/srf02/uart.c #####
```

```
#include <avr/interrupt.h>
#include <avr/io.h>
#define BAUD 12
#include "uart.h"

#define NULL (void*)0

/* UART variables */
static volatile uint8_t tx_read;
static volatile uint8_t tx_write;
static volatile uint8_t tx_buff[TX_BUFF_SIZE];

void uart_init()
{
  /* Enable the UART for sending and receiving */
  sbi(UCSRB, RXEN);
  sbi(UCSRB, TXEN);

  tx_read = tx_write = 0;

  /* Set baud rate registers */
  UBRRH = (((uint16_t)BAUD) & 0xff00) >> 8;
  UBRRL = ((uint16_t)BAUD) & 0xff;

  /* Enable Rx interrupt */
  sbi(UCSRB, RXCIE);
}

/* UART tx interrupt */
SIGNAL(SIG_UART_DATA)
{
  /* Tx buffer is empty */
  if(tx_read == tx_write)
  {
    cbi(UCSRB, UDRIE);
    return;
  }

  UDR = tx_buff[tx_read];

  tx_read = tx_read + 1 >= TX_BUFF_SIZE ? 0 : tx_read + 1;
}

void put_char(uint8_t c)
{
  uint8_t tmp_write;

  tmp_write = tx_write + 1 >= TX_BUFF_SIZE ? 0 : tx_write + 1;

  /* Buffer is full */
  if(tmp_write == tx_read)
  {
    sbi(UCSRB, UDRIE);
    return;
  }


  tx_buff[tx_write] = c;
  tx_write = tmp_write;

  /* Enable Tx interrupt to start sending */
  sbi(UCSRB, UDRIE);
```

```
  return;
}

void put_short(uint16_t s)
{
  put_char(((uint8_t)s & 0xFF));
  put_char((uint8_t)(s >> 8));
}

void put_str(char *str)
{
  uint32_t i = 0;

  while (str[i] != (char)NULL)
  {
    put_char(str[i]);
    i++;
  }
}


###### Subvehicle/Sensors/trunk/srf02/uart.h #####

#ifndef __MY_UART__
#define __MY_UART__

#include <avr/interrupt.h>
#include <avr/io.h>

#define TX_BUFF_SIZE 256

#ifndef BAUD
/* If different baud rate is desired do #define BAUD ??
   BEFORE #include "uart.h" in your source */
#define BAUD 51 /* 9600 baud @ 8MHz normal speed */
#endif

#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif

/* Initialize UART, MUST BE CALLED */
void uart_init();
/* Write 8-bit value to UART */
void put_char(uint8_t c);
/* Write 16-bit value to UART */
void put_short(uint16_t s);
/* Write string of length len to UART */
void put_str(char *str);

#endif /* __MY_UART__ */

###### Subvehicle/Sensors/trunk/srf02/i2c.h #####

#ifndef I2C__H__
#define I2C__H__

#include <avr/interrupt.h>
#include <avr/io.h>
```

```
#include <stdio.h>
#include <util/twi.h>
#define F_CPU 800000UL
#include <util/delay.h>

#define START_ERROR 2
#define ADDR_SENT_ERROR 3
#define DATA_SENT_ERROR 4
#define ADDR_REC_ERROR 5
#define DATA_REC_ERROR 6
#define DATA_NACK_REC_ERROR 7
#define SENT_OK 8
#define OK 0
#define ERROR 0xFF
#define START 0x08

extern uint8_t i2c_error;

void i2c_init();
int srf_change_addr(uint8_t curr_addr, uint8_t new_addr);
uint8_t srf_read(uint8_t addr);
int i2c_transmit(uint8_t addr, uint8_t data);
uint8_t i2c_read(uint8_t addr, uint8_t reg);
int send_repeat();
int send_start();
void send_stop();
int send_addr_t(uint8_t addr);
int send_addr_r(uint8_t addr);
int send_data(uint8_t data);
int get_data();

#endif //I2C__H__

###### Subvehicle/Sensors/trunk/GP2Y0A2/adc.h #####

#ifndef __MY_ADC__
#define __MY_ADC__

#include <avr/interrupt.h>
#include <avr/io.h>

void adc_init();
uint16_t read_adc();
#endif //__MY_ADC__

###### Subvehicle/Sensors/trunk/GP2Y0A2/main.c #####

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <avr/delay.h>

#include "uart.h"
#include "adc.h"

int main()
{

  uint16_t i, a[10], size = 0, p = 0, sum;
  char c[10];
  memset((void*)c,0,10);
```

```
  memset((void*)a, 0, 10*sizeof(uint16_t));

  /* Initialize UART peripherial, MUST BE CALLED */
  cli();
  uart_init();
  adc_init();
  sei();

 while(1){
 /* Delay so we don't send stuff too fast*/
    _delay_ms(32);
    _delay_ms(32);

    a[p] = read_adc();

    p = (p + 1) % 10;
    if (size < 10) size++;

    sum = 0;
    for (i= 0; i < size; i++) sum += a[i];

    snprintf(c, 10,"IR=%4d\r",(int)(sum/size));
    put_str(c,10);
  }
}

###### Subvehicle/Sensors/trunk/GP2Y0A2/test.c #####

#include <stdlib.h>
#include <stdio.h>

int main ( ) {
  int i = 45;
  char s[10];

  itoa(i, s, 10);

  printf("%s\n", s);

  return 0 ;
  }

###### Subvehicle/Sensors/trunk/GP2Y0A2/uart.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
#define BAUD 12
#include "uart.h"

/* UART variables */
static volatile uint8_t tx_read;
static volatile uint8_t tx_write;
static volatile uint8_t tx_buff[TX_BUFF_SIZE];

void uart_init()
{
  /* Enable the UART for sending and receiving */
  sbi(UCSRB, RXEN);
  sbi(UCSRB, TXEN);

  tx_read = tx_write = 0;

  /* Set baud rate registers */
```

```
  UBRRH = (((uint16_t)BAUD) & 0xff00) >> 8;
  UBRRL = ((uint16_t)BAUD) & 0xff;

  /* Enable Rx interrupt */
  sbi(UCSRB, RXCIE);
}

/* UART tx interrupt */
SIGNAL(SIG_UART_DATA)
{
  /* Tx buffer is empty */
  if(tx_read == tx_write)
  {
    cbi(UCSRB, UDRIE);
    return;
  }

  UDR = tx_buff[tx_read];

  tx_read = tx_read + 1 >= TX_BUFF_SIZE ? 0 : tx_read + 1;
}

void put_char(uint8_t c)
{
  uint8_t tmp_write;

  tmp_write = tx_write + 1 >= TX_BUFF_SIZE ? 0 : tx_write + 1;

  /* Buffer is full */
  if(tmp_write == tx_read)
  {
    sbi(UCSRB, UDRIE);
    return;
  }


  tx_buff[tx_write] = c;
  tx_write = tmp_write;

  /* Enable Tx interrupt to start sending */
  sbi(UCSRB, UDRIE);

  return;
}

void put_short(uint16_t s)
{
  put_char(((uint8_t)s & 0xFF));
  put_char((uint8_t)(s >> 8));
}

void put_str(char *str, uint32_t len)
{
  uint32_t i;

  for(i = 0; i < len; i++)
    put_char(str[i]);
}


###### Subvehicle/Sensors/trunk/GP2Y0A2/adc.c #####

#ifndef ADC__H__
```

```
#define ADC__H__

#include <avr/interrupt.h>
#include <avr/io.h>
#include "adc.h"

void adc_init(void)
{
  PORTA = 0x00; DDRA = 0xFE;
  ADMUX = 0xE0; //port0 enabled
  ADCSRA = 0x85;
}


uint16_t read_adc(void)
{
  ADCSRA |= (1 << ADSC);
  while( (ADCSRA & ADSC) >> ADSC);

  return (ADCH << 2);
}

#endif // ADC__H__

###### Subvehicle/Sensors/trunk/GP2Y0A2/uart.h #####

#ifndef __MY_UART__
#define __MY_UART__

#include <avr/interrupt.h>
#include <avr/io.h>

#define TX_BUFF_SIZE 256

#ifndef BAUD
/* If different baud rate is desired do #define BAUD ??
   BEFORE #include "uart.h" in your source */
#define BAUD 51 /* 9600 baud @ 8MHz normal speed */
#endif

#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif

/* Initialize UART, MUST BE CALLED */
void uart_init();
/* Write 8-bit value to UART */
void put_char(uint8_t c);
/* Write 16-bit value to UART */
void put_short(uint16_t s);
/* Write string of length len to UART */
void put_str(char *str, uint32_t len);

#endif /* __MY_UART__ */

###### Subvehicle/Sensors/trunk/HMC6352/i2c.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
```

```
#include "i2c.h"
#include "uart.h"


void i2c_init()
{
  //PORTC = 0x03;  //port0 and port1 internally pulled high
  //TWSR = (0 << TWPS0) & (0 << TWPS1); //prescalar 1 default 0
  TWBR = 8;//(F_CPU / 100000UL - 16) / 2; //0x02; //2 with prescalar 1 should = 100kHz
}//end init



int hmc_transmit(uint8_t addr, uint8_t data)
{
  //int status = 0;

  if ( send_start() == ERROR)
    return ERROR;
  if ( send_addr_t(addr) == ERROR)
    return ERROR;
  if (send_data(data) == ERROR)
    return ERROR;

  send_stop();


  return OK;
}//end i2c_transmit(addr, data);

uint16_t hmc_read(uint8_t addr)
{

  uint16_t data = 0;

  if (send_start() == ERROR)
    return ERROR;
  if (send_addr_r(addr) == ERROR)
    return ERROR;
  if (get_ack_data() == ERROR)
    return ERROR;

  data = TWDR << 8;

  if (get_nack_data() == ERROR)
    return ERROR;

  data += TWDR;

  send_stop();

  return data;
}//end i2c_read(addr, reg);



/******* send_repeat(); ***********************/
int send_repeat()
{
  TWCR = 0 | (1 << TWSTA) | (1 << TWINT) | (1 << TWEN); //repeat start code

  // wait for repeat start sent
  while ( !(TWCR & (1 << TWINT)));
```

```
  // check repeat start sent correct
  if ((TWSR & 0xF8) != TW_REP_START)
    return ERROR; //put_str("REP_START_ERROR\n\r");

  return OK;
}//end send_repeat()



/******* send_start() ******************************/
int send_start()
{
  /* send start */
  TWCR = 0 | (1 << TWINT) | (1 << TWSTA) | (1 << TWEN); //send start condition

  /* wait for start sent */
  while (! (TWCR & (1 << TWINT))); // wait for TWINT flag set.(indicates start sent.

  /* check start sent correct */
  if ((TWSR & 0xF8) != START)
    return ERROR; //put_str("START_ERROR\n\r");

  return OK;
}//end send_start



/****** send_addr_t(uint8_t addr); *********************/
int  send_addr_t(uint8_t addr)
{
  /* send addr */
  TWDR = addr;
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT to start transmission of
addresss.

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT)));

  /*check status = slave sent/ack received */
  if ((TWSR & 0xF8) != TW_MT_SLA_ACK)
    return ERROR;

  return OK;
}//end send_addr


/****** send_addr_r(uint8_t addr); *********************/
int send_addr_r(uint8_t addr)
{
  /* send addr */
  TWDR = addr | 0x01;

  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWEA); //clear TWINT to start
transmission of addresss. send ACK

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT set. indicates Slave addr sent
and ACK received.

  /*check status = slave sent/ack received */
  if ((TWSR & 0xF8) != TW_MR_SLA_ACK)
    return ERROR; // put_str("ADDR_SENT_ERROR\n\r");
```

```
  return OK;
}//end send_addr


/*************************************
 *      int send_data(uint8_t data);
 *************************************/
int send_data(uint8_t data)
{
  /* send data */
  TWDR = data;
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT bit to start data transmission.

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, ACK
received.

  /* check status = MT_DATA_ACK */
  if ( (TWSR & 0xF8) != TW_MT_DATA_ACK)
    return ERROR; // put_str("DATA_SENT_ERROR\n\r");

  return OK;
}// end send_data



/********** send_stop(); *****************/
void send_stop()
{
  /* send STOP condition */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);

  //return OK;
}



/********* int get_data(); ***************/
int get_data()
{
  /* get data */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT bit to start data read. send
NACK.

  /* wait for ACK sent */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, NACK sent.

  /* check status = MR_DATA_NACK */
  if ( (TWSR & 0xF8) != TW_MR_DATA_NACK)
    return ERROR; //put_str("DATA_REC_ERROR\n\r");

  return OK;
}
/********* int get_data(); ***************/
int get_nack_data()
{
  /* get data */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT bit to start data read. send
NACK.

  /* wait for ACK sent */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, NACK sent.
```

```
  /* check status = MR_DATA_NACK */
  if ( (TWSR & 0xF8) != TW_MR_DATA_NACK)
    return ERROR; //put_str("DATA_REC_ERROR\n\r");

  return OK;
}

/********* int get_data(); ***************/
int get_ack_data()
{
  /* get data */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWEA); //clear TWINT to start
transmission of addresss. send ACK

  /* wait for ACK sent */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, NACK sent.

  /* check status = MR_DATA_NACK */
  if ( (TWSR & 0xF8) != TW_MR_DATA_ACK)
    return ERROR; //put_str("DATA_REC_ERROR\n\r");

  return OK;
}
```

```
###### Subvehicle/Sensors/trunk/HMC6352/main.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
#define F_CPU 800000UL
#include "i2c.h"
#include "uart.h"

int main()
{
  char s[25];
  uint16_t range = 0;

  /* init uart and i2c */
  uart_init();
  i2c_init();
  sei();

  hmc_transmit(0x42, 0x57);
  _delay_ms(10);
  while(1)
  {
    _delay_ms(10000);
//    _delay_ms(100000);
    if ( hmc_transmit(0x42,0x41) != ERROR)
    {
      _delay_ms(100);
      if ( (range = hmc_read(0x42)) == ERROR)
      {
        send_stop();
      }
    }else
    {
      send_stop();
    }
```

```
    snprintf(s, 25, "d=%4d deg\r", range);
    put_str(s);
  }

 return(0);
}//end main


###### Subvehicle/Sensors/trunk/HMC6352/uart.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
#define BAUD 12
#include "uart.h"

#define NULL (void*)0

/* UART variables */
static volatile uint8_t tx_read;
static volatile uint8_t tx_write;
static volatile uint8_t tx_buff[TX_BUFF_SIZE];

void uart_init()
{
  /* Enable the UART for sending and receiving */
  sbi(UCSRB, RXEN);
  sbi(UCSRB, TXEN);

  tx_read = tx_write = 0;

  /* Set baud rate registers */
  UBRRH = (((uint16_t)BAUD) & 0xff00) >> 8;
  UBRRL = ((uint16_t)BAUD) & 0xff;

  /* Enable Rx interrupt */
  sbi(UCSRB, RXCIE);
}

/* UART tx interrupt */
SIGNAL(SIG_UART_DATA)
{
  /* Tx buffer is empty */
  if(tx_read == tx_write)
  {
    cbi(UCSRB, UDRIE);
    return;
  }

  UDR = tx_buff[tx_read];

  tx_read = tx_read + 1 >= TX_BUFF_SIZE ? 0 : tx_read + 1;
}

void put_char(uint8_t c)
{
  uint8_t tmp_write;

  tmp_write = tx_write + 1 >= TX_BUFF_SIZE ? 0 : tx_write + 1;

  /* Buffer is full */
  if(tmp_write == tx_read)
  {
```

```
    sbi(UCSRB, UDRIE);
    return;
  }


  tx_buff[tx_write] = c;
  tx_write = tmp_write;

  /* Enable Tx interrupt to start sending */
  sbi(UCSRB, UDRIE);

  return;
}

void put_short(uint16_t s)
{
  put_char(((uint8_t)s & 0xFF));
  put_char((uint8_t)(s >> 8));
}

void put_str(char *str)
{
  uint32_t i = 0;

  while (str[i] != (char)NULL)
  {
    put_char(str[i]);
    i++;
  }
}


###### Subvehicle/Sensors/trunk/HMC6352/uart.h #####

#ifndef __MY_UART__
#define __MY_UART__

#include <avr/interrupt.h>
#include <avr/io.h>

#define TX_BUFF_SIZE 256

#ifndef BAUD
/* If different baud rate is desired do #define BAUD ??
   BEFORE #include "uart.h" in your source */
#define BAUD 51 /* 9600 baud @ 8MHz normal speed */
#endif

#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif

/* Initialize UART, MUST BE CALLED */
void uart_init();
/* Write 8-bit value to UART */
void put_char(uint8_t c);
/* Write 16-bit value to UART */
void put_short(uint16_t s);
/* Write string of length len to UART */
```

```
void put_str(char *str);

#endif /* __MY_UART__ */

###### Subvehicle/Sensors/trunk/HMC6352/i2c.h #####

#ifndef I2C__H__
#define I2C__H__

#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>
#include <util/twi.h>
#define F_CPU 800000UL
#include <util/delay.h>

#define START_ERROR 1
#define ADDR_SENT_ERROR 2
#define DATA_SENT_ERROR 3
#define ADDR_REC_ERROR 4
#define DATA_REC_ERROR 5
#define DATA_NACK_REC_ERROR 6
#define SENT_OK 7
#define OK    0
#define ERROR 1
#define START 0x08

void i2c_init();
int hmc_transmit(uint8_t addr, uint8_t data);
uint16_t hmc_read(uint8_t addr);
int send_repeat();
int send_start();
void send_stop();
int send_addr_t(uint8_t addr);
int send_addr_r(uint8_t addr);
int send_data(uint8_t data);
int get_data();
int get_ack_data();
int get_nack_data();

#endif //I2C__H__

###### Subvehicle/Sensors/trunk/servo/main.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <util/delay.h>

#include "timer.h"
#include "uart.h"

int main()
{
  timer_init();
  uart_init();
  sei();
  timer_cnt=0;
  put_char('A');
  uint16_t i=650;
    char s[50]={0};
```

```
  int x=10;

  while(1)
  {


    move_servo(i);
    _delay_ms(32);

    i+=x;

    if (x > 0 && i + x > 2450) x = -x;
    else if (x < 0 && i + x < 650) x = -x;

    snprintf(s, 50,"i=%d\n\r", i);
    put_str(s);

  }

  return (0);
}

###### Subvehicle/Sensors/trunk/servo/uart.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
#define BAUD 12
#include "uart.h"

#define NULL (void*)0

/* UART variables */
static volatile uint8_t tx_read;
static volatile uint8_t tx_write;
static volatile uint8_t tx_buff[TX_BUFF_SIZE];
static volatile uint8_t rx_head;
static volatile uint8_t rx_tail;
static volatile uint8_t rx_buff[TX_BUFF_SIZE];

void uart_init()
{
  /* Enable the UART for sending and receiving */
  sbi(UCSRB, RXEN);
  sbi(UCSRB, TXEN);

  tx_read = tx_write = 0;

  /* Set baud rate registers */
  UBRRH = (((uint16_t)BAUD) & 0xff00) >> 8;
  UBRRL = ((uint16_t)BAUD) & 0xff;

  /* Enable Rx interrupt */
  sbi(UCSRB, RXCIE);
}


/*UART rc interrupt */
SIGNAL(SIG_UART_RECV)
{
  uint8_t tmp_head;

  tmp_head = rx_head + 1 % RX_BUFF_SIZE;
```

```
  /*if rx full*/
  while(tmp_head == rx_tail);

  rx_buff[rx_head] = UDR;

  rx_head = tmp_head;
}

uint8_t get_char()
{
  uint8_t tmp_tail;

  /*rx buffer empty*/
  if(rx_head == rx_tail)
    return((char)NULL);

  tmp_tail = rx_tail;
  rx_tail = (rx_tail + 1) % RX_BUFF_SIZE;

  return rx_buff[tmp_tail];
}

uint8_t get_str(char *buff, uint8_t size)
{
  uint32_t i = 0;

  while ( (buff[i] = get_char()) != (char)NULL && i < size)
    i++;

  return i;
}


/* UART tx interrupt */
SIGNAL(SIG_UART_DATA)
{
  /* Tx buffer is empty */
  if(tx_read == tx_write)
  {
    cbi(UCSRB, UDRIE);
    return;
  }

  UDR = tx_buff[tx_read];

  tx_read = tx_read + 1 >= TX_BUFF_SIZE ? 0 : tx_read + 1;
}

void put_char(uint8_t c)
{
  uint8_t tmp_write;

  tmp_write = tx_write + 1 >= TX_BUFF_SIZE ? 0 : tx_write + 1;

  /* Buffer is full */
  if(tmp_write == tx_read)
  {
    sbi(UCSRB, UDRIE);
    return;
  }


  tx_buff[tx_write] = c;
```

```
  tx_write = tmp_write;

  /* Enable Tx interrupt to start sending */
  sbi(UCSRB, UDRIE);

  return;
}

void put_short(uint16_t s)
{
  put_char(((uint8_t)s & 0xFF));
  put_char((uint8_t)(s >> 8));
}

void put_str(char *str)
{
  uint32_t i = 0;

  while (str[i] != (char)NULL)
  {
    put_char(str[i]);
    i++;
  }
}


###### Subvehicle/Sensors/trunk/servo/uart.h #####

#ifndef __MY_UART__
#define __MY_UART__

#include <avr/interrupt.h>
#include <avr/io.h>

#define TX_BUFF_SIZE 256
#define RX_BUFF_SIZE 256


#ifndef BAUD
/* If different baud rate is desired do #define BAUD ??
   BEFORE #include "uart.h" in your source */
#define BAUD 51 /* 38400 baud @ 8MHz normal speed */
#endif

#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif

/* Initialize UART, MUST BE CALLED */
void uart_init();
/* Write 8-bit value to UART */
void put_char(uint8_t c);
/* Write 16-bit value to UART */
void put_short(uint16_t s);
/* Write string to UART */
void put_str(char *str);
/* get 8-bit value from UART */
uint8_t get_char();
/* get string from UART */
```

```
uint8_t get_str(char*, uint8_t);

#endif /* __MY_UART__ */

###### Subvehicle/Sensors/trunk/servo/timer.h #####

#ifndef _timer_h_
#define _timer_h_

#define CLOCK ((unsigned) CLOCK_SPEED)

extern uint16_t timer_cnt;

void move_servo(uint16_t);
void timer_start();
void timer_stop();
void timer_reset();


void timer_init(void);


#endif //_timer_h_

###### Subvehicle/Sensors/trunk/servo/timer.c #####

#include <avr/interrupt.h>
#include "timer.h"

uint16_t timer_cnt;

void timer_init(void)
{
  DDRD |= (1<<5);  //enable PD5 as output for PWM signal
  TCCR1A = 0xA2;
  TCCR1B = 0x1A;
  ICR1= 50000;
  OCR1AH = 0x00;
  OCR1AL = 0x00;
  OCR1BH = 0x00;
  OCR1BL = 0x00;
  TIMSK = 0x04;  //enable timer overflow interrupt

  timer_cnt = 0;

}

void move_servo(uint16_t val)
{
  OCR1AH = (val & 0xFF00) >> 8;
  OCR1AL = val & 0x00FF;
}

void timer_start()
{

  TCCR1A = 0x0C;  //normal mode
  TCCR1B = 0x05;  //set freq to

}

void timer_stop()
{
```

```
  TCCR1A = 0X00;
  TCCR1B = 0X00;

}

void timer_reset()
{
  TCNT1H = 0x00;
  TCNT1L = 0x00;
  timer_cnt = 0;

}

ISR(TIMER1_OVF_vect)
{
  timer_cnt++;
}
```

Subvehicle/Motorcontroller/adc.h

```
#ifndef __MY_ADC__
#define __MY_ADC__

#include <avr/interrupt.h>
#include <avr/io.h>

extern uint8_t lval, uval;

void adc_init();
uint16_t read_adc(void);

#endif //__MY_ADC__
```

Subvehicle/Motorcontroller/MControl.c

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include "MControl.h"
#include <util/delay.h>
#include "i2c.h"



#define gain_prop 1
#define gain_deriv 1
#define gain_sensor 1

uint16_t heading;
uint16_t heading_error = 0;
uint16_t prev_heading_error = 0;


/*********************
 *
 * Go Foward Function
 * Sets INA for both H-Bridges to 1
 * and sets INB for both to 0
 *
 *********************/

void go_forward ( )
```

```
{

  PORTC |= (1<<3) | (1<<5);
  PORTC &= ~(1<<2) & ~(1<<4);

  return;
}


/**********************
 *
 * Go Reverse Function
 * Sets INA for both H-Bridges to 0
 * and sets INB for both to 1
 *
 **********************/

void go_reverse ( )
{
  PORTC |= (1<<2) | (1<<4);
  PORTC &= ~(1<<3) & ~(1<<5);

  return;
}


/**********************
 *
 * Turn Left Function
 * Makes 4,5 go reverse
 * Makes 6,7 go forward
 *
 **********************/

void turn_left ( )
{
  PORTC |= (1<<3) | (1<<4);
  PORTC &= ~(1<<2) & ~(1<<5);

  return;
}

/**********************
 *
 * Turn Right Function
 * Makes 4,5 go forward
 * Makes 6,7 go reverse
 *
 **********************/

void turn_right ( )
{
  PORTC |= (1<<2) | (1<<5);
  PORTC &= ~(1<<3) & ~(1<<4);

  return;
}

/**********************
 *
 * Brake Function
 *
 **********************/
```

```
void brake ( )
{
  PORTC &= ~(1<<2) & ~(1<<3) & ~(1<<5) & ~(1<<4);
  _delay_ms(10);

  return;
}

/********************
 *
 * Update PID function
 *
 * *****************/

uint16_t update_PID(uint16_t heading_desired)
{
  uint16_t pid;
  uint16_t P = 0;
  uint16_t D = 0;
  int dt;

  dt = (timer_cnt * 65535) + ((TCNT1H << 8) + TCNT1L);

  heading = get_compass(0x42);

  //calculate heading error for proportional
  heading_error = heading_desired - heading;

  P = gain_prop * heading_error;
  if (dt != 0) D = gain_deriv * (heading_error-prev_heading_error)/dt;
  pid = P + D;

  prev_heading_error = heading_error;
  //timestamp = prev_timestamp;

  return pid;
}


###### Subvehicle/Motorcontroller/MControl.h #####

#ifndef MControl__h__
#define MControl__h__

#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/delay.h>

void go_forward();
void go_reverse();
void turn_left();
void turn_right();
void brake();
uint16_t update_PID(uint16_t heading_desired);


#endif

###### Subvehicle/Motorcontroller/i2c.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
```

```
#include "i2c.h"
//#include "uart.h"

#define nop() _delay_us(10)

void i2c_init()
{
  //PORTC = 0x03;  //port0 and port1 internally pulled high
  //TWSR = (0 << TWPS0) & (0 << TWPS1); //prescalar 1 default 0
  TWBR = 8;//(F_CPU / 100000UL - 16) / 2; //0x02; //2 with prescalar 1 should = 100kHz
//  hmc_transmit(0x42, 0x57);
//  i2c_transmit(0x3A, 0x20, 0xB7);
}//end init


uint16_t get_compass(uint8_t addr)
{
  uint8_t check = 0;
  check = hmc_transmit(addr, 0x41);
  _delay_ms(7);
 return (hmc_read(addr));
}
void get_accel(uint8_t addr, uint16_t *data)
{
  data[0] = i2c_read(addr, 0x28);
  nop();
  data[0] += i2c_read(addr, 0x29) << 8;
  nop();
  data[1] = i2c_read(addr, 0x2A);
  nop();
  data[1] += i2c_read(addr, 0x2B) << 8;
  nop();
  data[2] = i2c_read(addr, 0x2C);
  nop();
  data[2] += i2c_read(addr, 0x2D) << 8;
}

uint16_t get_sonar(uint8_t addr)
{
    uint8_t check = 0;
    uint16_t range = 0;
    check = i2c_transmit(addr,0,0x51);
    range = i2c_read(addr,2);
    range = range << 8;
    range += i2c_read(addr,3);
    return(range);
}

uint8_t i2c_transmit(uint8_t addr, uint8_t reg, uint8_t data)
{
  if ( send_start() == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if ( send_addr_t(addr) == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if ( send_data(reg) == ERROR)
  {
    send_stop();
```

```
    return ERROR;
  }
  if (send_data(data) == ERROR)
  {
    send_stop();
    return ERROR;
  }

  send_stop();
  return OK;
}//end i2c_transmit(addr, data);


uint8_t i2c_read(uint8_t addr, uint8_t reg)
{
  uint8_t data = 0;

  if (send_start() ==ERROR)
  {
    send_stop();
    return ERROR;
  }
  if (send_addr_t(addr) == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if (send_data(reg) == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if (send_repeat() == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if (send_addr_r(addr) == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if (get_nack_data() == ERROR)
  {
    send_stop();
    return ERROR;
  }

  data = TWDR;
  send_stop();
  return data;
}//end i2c_read(addr, reg);

uint8_t hmc_transmit(uint8_t addr, uint8_t data)
{
  //int status = 0;

  if ( send_start() == ERROR)
  {
    send_stop();
    return ERROR;
  }
```

```
  if ( send_addr_t(addr) == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if (send_data(data) == ERROR)
  {
    send_stop();
    return ERROR;
  }

  send_stop();
  return OK;
}//end i2c_transmit(addr, data);

uint16_t hmc_read(uint8_t addr)
{

  uint16_t data = 0;

  if (send_start() == ERROR)
  {
    send_stop();
    return ERROR;
  }

  if (send_addr_r(addr) == ERROR)
  {
    send_stop();
    return ERROR;
  }

  if (get_ack_data() == ERROR)
  {
    send_stop();
    return ERROR;
  }

  data = TWDR << 8;

  if (get_nack_data() == ERROR)
  {
    send_stop();
    return ERROR;
  }

  data += TWDR;
  send_stop();
  return data;
}//end hmc_read(addr, reg);



/******* send_repeat(); ***********************/
int send_repeat()
{
  TWCR = 0 | (1 << TWSTA) | (1 << TWINT) | (1 << TWEN); //repeat start code

  // wait for repeat start sent
  while ( !(TWCR & (1 << TWINT)));

  // check repeat start sent correct
  if ((TWSR & 0xF8) != TW_REP_START)
```

```
    return ERROR; //put_str("REP_START_ERROR\n\r");

  return OK;
}//end send_repeat()



/******* send_start() ******************************/
int send_start()
{
  /* send start */
  TWCR = 0 | (1 << TWINT) | (1 << TWSTA) | (1 << TWEN); //send start condition

  /* wait for start sent */
  while (! (TWCR & (1 << TWINT))); // wait for TWINT flag set.(indicates start sent.

  /* check start sent correct */
  if ((TWSR & 0xF8) != START)
    return ERROR; //put_str("START_ERROR\n\r");

  return OK;
}//end send_start



/****** send_addr_t(uint8_t addr); *********************/
int  send_addr_t(uint8_t addr)
{
  /* send addr */
  TWDR = addr;
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT to start transmission of
addresss.

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT)));

  /*check status = slave sent/ack received */
  if ((TWSR & 0xF8) != TW_MT_SLA_ACK)
    return ERROR;

  return OK;
}//end send_addr


/****** send_addr_r(uint8_t addr); *********************/
int send_addr_r(uint8_t addr)
{
  /* send addr */
  TWDR = addr | 0x01;

  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWEA); //clear TWINT to start
transmission of addresss. send ACK

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT set. indicates Slave addr sent
and ACK received.

  /*check status = slave sent/ack received */
  if ((TWSR & 0xF8) != TW_MR_SLA_ACK)
    return ERROR; // put_str("ADDR_SENT_ERROR\n\r");

  return OK;
}//end send_addr
```

```
/*************************************
 *      int send_data(uint8_t data);
 *************************************/
int send_data(uint8_t data)
{
  /* send data */
  TWDR = data;
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT bit to start data transmission.

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, ACK
received.

  /* check status = MT_DATA_ACK */
  if ( (TWSR & 0xF8) != TW_MT_DATA_ACK)
    return ERROR; // put_str("DATA_SENT_ERROR\n\r");

  return OK;
}// end send_data



/********** send_stop(); *****************/
void send_stop()
{
  /* send STOP condition */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);

  //return OK;
}



/********* int get_data(); ***************/
int get_data()
{
  /* get data */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT bit to start data read. send
NACK.

  /* wait for ACK sent */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, NACK sent.

  /* check status = MR_DATA_NACK */
  if ( (TWSR & 0xF8) != TW_MR_DATA_NACK)
    return ERROR; //put_str("DATA_REC_ERROR\n\r");

  return OK;
}
/********* int get_data(); ***************/
int get_nack_data()
{
  /* get data */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT bit to start data read. send
NACK.

  /* wait for ACK sent */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, NACK sent.

  /* check status = MR_DATA_NACK */
  if ( (TWSR & 0xF8) != TW_MR_DATA_NACK)
```

```
    return ERROR; //put_str("DATA_REC_ERROR\n\r");

  return OK;
}

/********* int get_data(); ***************/
int get_ack_data()
{
  /* get data */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWEA); //clear TWINT to start
transmission of addresss. send ACK

  /* wait for ACK sent */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, NACK sent.

  /* check status = MR_DATA_NACK */
  if ( (TWSR & 0xF8) != TW_MR_DATA_ACK)
    return ERROR; //put_str("DATA_REC_ERROR\n\r");

  return OK;
}
```

Subvehicle/Motorcontroller/main.c

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include "timer.h"
#include <util/delay.h>
#include "MControl.h"
#include "i2c.h"
#include "uart.h"
#include <avr/wdt.h>

int main()
{
  int i = 0;
  int j = 0;
  char s[100];
  uint16_t accel_data[3] = {0};

  timer_init();
  uart_init();
  sei();
  wdt_disable();

  DDRC = 0xFF;     //enable PC2-7 as logic level outputs for motorcontroller
  DDRB |= (1<<3);  //enable PB3 as output for PWM signal
  DDRD |= (1<<7);  //enable PD7 as output for PWM signal

  //hmc_transmit(0x42, 0x57);
  //i2c_transmit(0x3A, 0x20, 0xB7);

  brake();

  //snprintf(s,60, "Heading = %5u\n\n\r", (get_compass(0x42)));
  //put_str(s);

  timer_start();
  go_reverse();
```

```
  while(timer_cnt <= 2)
  {
    for (i = 0;i<2;i++) _delay_ms(32);
    //get_accel(0x3A, accel_data);
    snprintf(s, 100, "timer_cnt = %4u timer = %5u SonarE0 = %5u SonarE4 = %5u SonarE6
= %5u\r",timer_cnt,(TCNT1H << 8) +TCNT1L, (get_compass(0x42)), get_sonar(0xE0),
get_sonar(0xE4), get_sonar(0xE6));
    put_str(s);

    if ((get_sonar(0xE6)) < 278)
    {
      brake();
      turn_left();
      for(j = 0;j<32;j++) _delay_ms(32);
      brake();
      go_reverse();
    }

  }

  brake();
  timer_stop();

  put_str("\nDone!!\n\r");

  /*go_forward();
  for (i = 0;i<63;i++) _delay_ms(32);  //delay for 5 seconds

  brake();
  for (i = 0;i<32;i++) _delay_ms(32);  //delay for 1 seconds

  go_reverse();
  for (i = 0;i<63;i++) _delay_ms(32);  //delay for 5 seconds

  brake();
  for (i = 0;i<32;i++) _delay_ms(32);  //delay for 1 seconds

  turn_left();
  for (i = 0;i<63;i++) _delay_ms(32);  //delay for 2 seconds

  brake();
  for (i = 0;i<32;i++) _delay_ms(32);  //delay for 1 seconds

  turn_right();
  for (i = 0;i<63;i++) _delay_ms(32);  //delay for 2 seconds
*/
  brake();

  return 0;

}
```

```
###### Subvehicle/Motorcontroller/uart.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
#define BAUD 12
#include "uart.h"

#define NULL (void*)0
```

```
/* UART variables */
static volatile uint8_t tx_read;
static volatile uint8_t tx_write;
static volatile uint8_t tx_buff[TX_BUFF_SIZE];

void uart_init()
{
  /* Enable the UART for sending and receiving */
  sbi(UCSRB, RXEN);
  sbi(UCSRB, TXEN);

  tx_read = tx_write = 0;

  /* Set baud rate registers */
  UBRRH = (((uint16_t)BAUD) & 0xff00) >> 8;
  UBRRL = ((uint16_t)BAUD) & 0xff;

  /* Enable Rx interrupt */
  sbi(UCSRB, RXCIE);
}

/* UART tx interrupt */
SIGNAL(SIG_UART_DATA)
{
  /* Tx buffer is empty */
  if(tx_read == tx_write)
  {
    cbi(UCSRB, UDRIE);
    return;
  }

  UDR = tx_buff[tx_read];

  tx_read = tx_read + 1 >= TX_BUFF_SIZE ? 0 : tx_read + 1;
}

void put_char(uint8_t c)
{
  uint8_t tmp_write;

  tmp_write = tx_write + 1 >= TX_BUFF_SIZE ? 0 : tx_write + 1;

  /* Buffer is full */
  if(tmp_write == tx_read)
  {
    sbi(UCSRB, UDRIE);
    return;
  }


  tx_buff[tx_write] = c;
  tx_write = tmp_write;

  /* Enable Tx interrupt to start sending */
  sbi(UCSRB, UDRIE);

  return;
}

void put_short(uint16_t s)
{
  put_char(((uint8_t)s & 0xFF));
```

```
  put_char((uint8_t)(s >> 8));
}

void put_str(char *str)
{
  uint32_t i = 0;

  while (str[i] != (char)NULL)
  {
    put_char(str[i]);
    i++;
  }
}


###### Subvehicle/Motorcontroller/adc.c #####

#ifndef ADC__H__
#define ADC__H__

#include <avr/interrupt.h>
#include <avr/io.h>
#include "adc.h"

uint8_t uval, lval;

void adc_init(void)
{
  ADCSRA = 0x00;
  PORTA = 0x00; DDRA = 0xFE; //port a is input others are output
  ADMUX = 0xC0; //port0 enabled
  ADCSRA = 0x83; //enabled.start.auto trigger.interrupt flag. interupt
enabled.prescalars
 // SFIOR = 0x00; //free running mode, set auto trigger to interupt flag
 // ADCSRA |= (1 << ADEN);
 // ADCSRA |= (1 << ADSC);
 // return;
}

void adc_isr(void)
{
  lval = (uint8_t)ADCL;
  uval = (uint8_t)ADCH;
  return;
}

uint16_t read_adc(void)
{
  uint16_t i;
  uint8_t x;
  ADCSRA |= (1 << ADSC);
  while( (ADCSRA & ADSC) >> ADSC);

  x = ADCL;
  i = 0x0003 & ADCH;
  i <<= 8;
  i = i | (0x00FF & x);
  return i;
}

#endif // ADC__H__

###### Subvehicle/Motorcontroller/uart.h #####
```

```
#ifndef __MY_UART__
#define __MY_UART__

#include <avr/interrupt.h>
#include <avr/io.h>

#define TX_BUFF_SIZE 256

#ifndef BAUD
/* If different baud rate is desired do #define BAUD ??
   BEFORE #include "uart.h" in your source */
#define BAUD 51 /* 9600 baud @ 8MHz normal speed */
#endif

#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif

/* Initialize UART, MUST BE CALLED */
void uart_init();
/* Write 8-bit value to UART */
void put_char(uint8_t c);
/* Write 16-bit value to UART */
void put_short(uint16_t s);
/* Write string of length len to UART */
void put_str(char *str);

#endif /* __MY_UART__ */

###### Subvehicle/Motorcontroller/timer.h #####

#ifndef _timer_h_
#define _timer_h_

#define CLOCK ((unsigned) CLOCK_SPEED)

extern uint16_t timer_cnt;

void timer_start();
void timer_stop();
void timer_reset();


inline void timer_init(void);


#endif //_timer_h_

###### Subvehicle/Motorcontroller/timer.c #####

#include <avr/interrupt.h>
#include "timer.h"

uint16_t timer_cnt;

void timer_init(void)
{
         //Timer0 @ clk0: Motor Controller
```

```
          TCCR0 = 0x6A;  //Set Timer/Counter0 to fast PWM mode
                         //Set output compare to asserted state
                         //Prescaler factor to set PWM freq at 3.9KHz
                          //max PWM freq is 20KHz
          OCR0 = 0x5F;   //set output compare register to ??

          TCCR2 = 0x6A;  //Set Timer/Counter2 to fast PWM mode
                          //Set output compare to asserted state
                          //Prescaler factor to set PWM at 3.9KHz
          OCR2 = 0x5F;   //Set OC reg to something

  TIMSK = 0x04;  //enable timer overflow interrupt

  timer_cnt = 0;

}

void timer_start()
{

  TCCR1A = 0x0C;  //normal mode
  TCCR1B = 0x05;  //set freq to

}

void timer_stop()
{

  TCCR1A = 0X00;
  TCCR1B = 0X00;

}

void timer_reset()
{
  TCNT1H = 0x00;
  TCNT1L = 0x00;
  timer_cnt = 0;

}

ISR(TIMER1_OVF_vect)
{

  timer_cnt++;

}


###### Subvehicle/Motorcontroller/i2c.h #####

#ifndef I2C__H__
#define I2C__H__

#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>
#include <util/twi.h>
//#define F_CPU 800000UL
#include <util/delay.h>

#define START_ERROR 1
#define ADDR_SENT_ERROR 2
```

```
#define DATA_SENT_ERROR 3
#define ADDR_REC_ERROR 4
#define DATA_REC_ERROR 5
#define DATA_NACK_REC_ERROR 6
#define SENT_OK 7
#define OK    0
#define ERROR 1
#define START 0x08

void i2c_init();

uint16_t get_compass(uint8_t addr);
void get_accel(uint8_t addr, uint16_t *data);
uint16_t get_sonar(uint8_t addr);

uint8_t hmc_transmit(uint8_t addr, uint8_t data);
uint16_t hmc_read(uint8_t addr);
uint8_t i2c_transmit(uint8_t addr, uint8_t reg, uint8_t data);
uint8_t i2c_read(uint8_t addr, uint8_t reg);

/******/
int send_repeat();
int send_start();
void send_stop();
int send_addr_t(uint8_t addr);
int send_addr_r(uint8_t addr);
int send_data(uint8_t data);
int get_data();
int get_ack_data();
int get_nack_data();
/*****/
#endif //I2C__H__

###### Subvehicle/refs/od/od_LoadImage_test.c #####

/* od_LoadImage_test.c
 *
 * ObjectDetection: odLoadImage( ) test file
 */


int main( )
{
  return 0;
} /* main */

/* eof */


###### Subvehicle/refs/od/camtest.c #####

/* camtest.c
 *
 */

#include <sys/time.h>

#include "od_Camera.h"


int main( )
{
  struct timeval time;
```

```
  char *image;
  int shutter_speed = 52000, gain = 33000;
  // SOMEONE NEEDS TO LOOK AT THE SHUTTER SPEED AND GAIN VARIABLES

  odCameraMove( 6000, -4500 );
  image = odCameraTakePicture( &time, shutter_speed, gain );
  odCameraImageToJPEGFile( image, "/root/bin/test.jpg", 100 );

  return 0;
} /* main */

/* eof */
```

Subvehicle/refs/od/od_ImageProcess.h

```
/* od_ImageProcess.h
 *
 * ObjectDetection: Handling each image using OpenCV
 *
 */


#ifndef __OD_IMAGEPROCESS__H__
#define __OD_IMAGEPROCESS__H__


#define OD_WIN_SOURCE       "Source Image"
#define OD_WIN_FINAL        "Final Image"
#define OD_WIN_3            "Process 1"
#define OD_WIN_4            "Process 2"


#define INF                 1000.0f

#include <opencv/cv.h>
#include <math.h>

#include "od_common.h"
#include "od_dbase.h"


#ifdef HAVE_OPENCV_HIGHGUI_H

#include <opencv/highgui.h>

int odCreateWindows( );
void odDestroyWindows( );
void odShowWindows( IplImage *source, IplImage *final );
int odDBtoCvSeq_Overview( od_DB *db, CvSeq **flines, CvMemStorage *fline_storage );

#endif /* HAVE_OPENCV_HIGHGUI_H */

IplImage *odLoadImage( void *src );
IplImage *odAnalyzeImage( IplImage *source, float thresh, od_DB *maindb );
int odAnalyzeLines( CvSeq *lines, od_DB *db );
float __inline odVectorLength2D( CvPoint *line );

float odVectorAngle2D( CvPoint *lineA, CvPoint *lineB );


#endif /* __OD_IMAGEPROCESS__H__ */
```

```
/* eof */

###### Subvehicle/refs/od/od_Network.h #####

/* od_Network.h
 *
 * ObjectDetection:
 *
 *
 */

#ifndef __OD_NETWORK__H__
#define __OD_NETWORK__H__

#include "od_common.h"


#define OD_NET_MAX_CLIENTS      4

#define OD_NET_PACKET_NEW       1
#define OD_NET_PACKET_CONT      2


typedef struct _od_Packet {
  int size;     // The complete size of the packet
  int command;  // Any command given

  char *buffer; // Data sent
} od_Packet;


#endif /* OD_NETWORK__H__ */

/* eof */


###### Subvehicle/refs/od/od_NetLAN.c #####

/* od_NetLAN.c
 *
 * ObjectDetection:
 *
 */

#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <errno.h>
#include <sys/poll.h>
#include <sys/signal.h>
#include <fcntl.h>
#include <unistd.h>

#include "od_NetLAN.h"


static pthread_mutex_t server_lock = PTHREAD_MUTEX_INITIALIZER;
static od_LANServer *server = NULL;
static int go = 0;


int odCreateLANServer( )
```

```
{
  int port = OD_NET_LAN_PORT;
  int listen_socket, err;

  struct sockaddr_in  server_sockaddr;

  listen_socket = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );
  if( listen_socket == -1 ) {
    printf( "error in socket( )\n" );
    return OD_ERROR;
  }

  server_sockaddr.sin_family = AF_INET;
  server_sockaddr.sin_addr.s_addr = INADDR_ANY;
  server_sockaddr.sin_port = htons( port );

  err = bind( listen_socket, (struct sockaddr *)&server_sockaddr, sizeof( struct
sockaddr_in ));
  if( err == -1 ) {
    printf( "error in bind( )\n" );
    close( listen_socket );
    return OD_ERROR;
  }

  err = listen( listen_socket, SOMAXCONN );
  if( err == -1 ) {
    printf( "error in listen( )\n" );
    close( listen_socket );
    return OD_ERROR;
  }

  go = 1;
  server = (od_LANServer *)malloc( sizeof( od_LANServer ));
  assert( server );

  server->listen_socket = listen_socket;

  fcntl( server->listen_socket, F_SETFL, O_NONBLOCK );
  if( 0 != pthread_create( &(server->sthandle), NULL, odLANServer_thread, (void *)NULL
)) {
    printf( "error in pthread_create( )\n" );
    server->listen_socket = 0;
    close( listen_socket );
    free( server );
    return OD_ERROR;
  }

  return OD_OK;
} /* odCreateLANServer */


int odDestroyLANServer( )
{
  int sflags, err;

  assert( server );
  go = 0;  pthread_join( server->sthandle, NULL );
  pthread_join( server->sthandle, NULL );

  return OD_OK;
} /* odDestroyLANServer */
```

```
void _sigiohandler( int arg )
{
  printf( "sigio called\n" );
} /* sigiohandler */


void *odLANServer_thread( void *args )
{
  int accept_socket;
  int i;


  server->num_clients = 0;
  memset( server->clients, 0, sizeof( pthread_t ) * OD_NET_MAX_CLIENTS );

  while( go ) {
    accept_socket = accept( server->listen_socket, NULL, NULL );
    if( accept_socket == -1 ) {
      if( errno != EAGAIN ) printf( "Error accept( ) errno = %d\n", errno );
      else sleep( 2 );
      continue;
    }

    if( server->num_clients >= OD_NET_MAX_CLIENTS ) {
      printf( "Error, too many clients\n" );
      close( accept_socket );
      continue;
    }
    else {
      for( i = 0; i < OD_NET_MAX_CLIENTS; i++ )
        if( server->clients[i] == 0 ) break;

      if( 0 != pthread_create( &(server->clients[i]), NULL, odLANClient_thread, (void
*)i )) {
        server->clients[i] = 0;
        printf( "Error creating new client thread\n" );
        close( accept_socket );
        continue;
      }
      else {
        pthread_mutex_lock( &server_lock );
        server->num_clients++;
        server->clients_socks[i] = accept_socket;
        pthread_mutex_unlock( &server_lock );
      }
    }

  } /* while( go ) */

  pthread_mutex_lock( &server_lock );
  for( i = 0; i < OD_NET_MAX_CLIENTS; i++ )
    if( server->clients[i] != 0 ) {
      server->clients_state[i] = 0;
      // send quit message
      close( server->clients_socks[i] );
      server->clients_socks[i] = 0;
    }
  pthread_mutex_unlock( &server_lock );

  close( server->listen_socket );
  free( server );
} /* odLANServer_thread */
```

```
void *odLANClient_thread( void *args )
{
  int client_number = (int)args;
  int err;
  int cpos, contpos, packet_mode = OD_NET_PACKET_NEW;
  char *buffer, *msg;
  int msg_size;

  buffer = (char *)malloc( sizeof( char ) * OD_NET_BUFFER_SIZE );
  assert( buffer );

  pthread_mutex_lock( &server_lock );
  server->clients_state[client_number] = 1;
  pthread_mutex_unlock( &server_lock );

  while( server->clients_state[client_number] ) {
    err == recv( server->clients_socks[client_number], buffer, OD_NET_BUFFER_SIZE, 0
);
    if( err == -1 ) {
      printf( "Error on recv( )\n" );
      continue;
    }

    cpos = 0;
    while( cpos < OD_NET_BUFFER_SIZE ) {

      if( cpos >= OD_NET_BUFFER_SIZE - 4 ) {

      }
      else {
        msg_size = ((od_Packet *)&buffer[cpos])->size;
      }

      switch( packet_mode ) {
        case OD_NET_PACKET_NEW:
          msg = (char *)malloc( msg_size );
          if( msg_size > ( OD_NET_BUFFER_SIZE - cpos )) {

            packet_mode = OD_NET_PACKET_CONT;
          }
          else {
          }
          break;
        case OD_NET_PACKET_CONT:

          break;
        default:
          printf( "This should not have been reached in odLANClient_thread( )\n" );
          break;
      }
    } /* while( 1 ) */
  }

  // send quit message
  close( server->clients_socks[client_number] );
  pthread_mutex_lock( &server_lock );
  server->clients_socks[client_number] = 0;
  server->num_clients--;
  server->clients[client_number] = 0;
  pthread_mutex_unlock( &server_lock );
} /* odLANDClient_thread */
```

```
int odLANSend( char *buffer, int size, int client_id )
{
  assert(( client_id >= 0 && client_id ) < ( OD_NET_MAX_CLIENTS ));
  assert( server->clients_socks[client_id] != 0 );

  send( server->clients_socks[client_id], buffer, size, 0 );

  return OD_OK;
} /* odLANSend */


int main( )
{
  odCreateLANServer( );

  sleep( 1 );

  odDestroyLANServer( );

  return 0;
}

/* eof */


###### Subvehicle/refs/od/od_Camera.c #####

/*
 * od_Camera.c
 *
 * Many of the functions are from vidcat.c from the w3cam project
 *
 * Copyright (C) 1998 - 2001 Rasca, Berlin
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/types.h>
#include <linux/videodev.h>
#include <jpeglib.h>

#include "od_common.h"
```

```
#include "od_Camera.h"
#include "otherlibs/pwc-ioctl.h"
#include "otherlibs/v4l.h"
#include "otherlibs/philipscamera.h"



#define DEF_WIDTH  640      /* default width */
#define DEF_HEIGHT 480      /* default height */

#define VIDEO_DEV  "/dev/v4l/video0"


/*
* read rgb image from v4l device
* return: mmap'ed buffer and size
*/
char *
get_image (int dev, int width, int height, int *size)
{
  struct video_mbuf vid_buf;
  struct video_mmap vid_mmap;
  char *map, *convmap;
  int len;
  int bytes = 3;

  if (ioctl (dev, VIDIOCGMBUF, &vid_buf) == -1) {
    /* to do a normal read()
    */
    struct video_window vid_win;

    if (ioctl (dev, VIDIOCGWIN, &vid_win) != -1) {
      vid_win.width  = width;
      vid_win.height = height;
      if (ioctl (dev, VIDIOCSWIN, &vid_win) == -1) {
        perror ("ioctl(VIDIOCSWIN)");
        return (NULL);
      }
    }

    map = malloc (width * height * bytes);
    len = read (dev, map, width * height * bytes);
    if (len <=  0) {
      free (map);
      return (NULL);
    }
    *size = 0;

    convmap = malloc ( width * height * bytes );
    v4l_yuv420p2rgb (convmap, map, width, height, bytes * 8);
    memcpy (map, convmap, (size_t) width * height * bytes);
    free (convmap);

    return (map);
  }

  map = mmap (0, vid_buf.size, PROT_READ|PROT_WRITE,MAP_SHARED,dev,0);
  if ((unsigned char *)-1 == (unsigned char *)map) {
    perror ("mmap()");
    return (NULL);
  }

  vid_mmap.format = VIDEO_PALETTE_YUV420P;
```

```
  vid_mmap.frame = 0;
  vid_mmap.width = width;
  vid_mmap.height = height;
  if (ioctl (dev, VIDIOCMCAPTURE, &vid_mmap) == -1) {
    perror ("VIDIOCMCAPTURE");
    fprintf (stderr, "args: width=%d height=%d palette=%d\n",
      vid_mmap.width, vid_mmap.height, vid_mmap.format);
    munmap (map, vid_buf.size);
    return (NULL);
  }
  if (ioctl (dev, VIDIOCSYNC, &vid_mmap.frame) == -1) {
    perror ("VIDIOCSYNC");
    munmap (map, vid_buf.size);
    return (NULL);
  }
  *size = vid_buf.size;

  convmap = malloc ( width * height * bytes );
  v4l_yuv420p2rgb (convmap, map, width, height, bytes * 8);
  memcpy (map, convmap, (size_t) width * height * bytes);
  free (convmap);

  return (map);
}

/*
* write ppm image to stdout / file
*/
void
put_image_ppm (FILE *out, char *image, int width, int height )
{
  int x;
  unsigned char *p = (unsigned char *)image;
  unsigned char buff[3];

  fprintf (out, "P6\n%d %d\n%d\n", width, height, 255);
  for (x = 0; x < width * height; x++) {
    buff[0] = p[2];
    buff[1] = p[1];
    buff[2] = p[0];
    fwrite (buff, 1, 3, out);
    p += 3;
  }

  fflush (out);
}

/*
*/
void
put_image_jpeg (FILE *out, char *image, int width, int height, int quality, int
palette)
{
  int y, x, line_width;
  JSAMPROW row_ptr[1];
  struct jpeg_compress_struct cjpeg;
  struct jpeg_error_mgr jerr;
  char *line;

  line = malloc (width * 3);
  if (!line)
    return;
  cjpeg.err = jpeg_std_error(&jerr);
```

```
  jpeg_create_compress (&cjpeg);
  cjpeg.image_width = width;
  cjpeg.image_height= height;

  cjpeg.input_components = 3;
  cjpeg.in_color_space = JCS_RGB;

  jpeg_set_defaults (&cjpeg);
  jpeg_set_quality (&cjpeg, quality, TRUE);
  cjpeg.dct_method = JDCT_FASTEST;
  jpeg_stdio_dest (&cjpeg, out);


  jpeg_start_compress (&cjpeg, TRUE);
  row_ptr[0] = line;
  if (palette == VIDEO_PALETTE_GREY) {
    line_width = width;
    for ( y = 0; y < height; y++) {
      row_ptr[0] = image;
      jpeg_write_scanlines (&cjpeg, row_ptr, 1);
      image += line_width;
    }
  } else {
    line_width = width * 3;
    for ( y = 0; y < height; y++) {
      for (x = 0; x < line_width; x+=3) {
        line[x]   = image[x+2];
        line[x+1] = image[x+1];
        line[x+2] = image[x];
      }
      jpeg_write_scanlines (&cjpeg, row_ptr, 1);
      image += line_width;
    }
  }
  jpeg_finish_compress (&cjpeg);
  jpeg_destroy_compress (&cjpeg);
  free (line);
}


int odCameraImageToJPEGFile( char *image, char *filename, int quality )
{
  FILE *out;

  out = fopen( filename, "wb" );
  if( !out ) return OD_ERROR;
  put_image_jpeg( out, image, OD_PIC_WIDTH, OD_PIC_HEIGHT, quality,
VIDEO_PALETTE_YUV420P );
  fclose( out );

  return OD_OK;
} /* odCameraImageToJPEGFile */


static int csize;

int odCameraImageToFile( char *image, char *filename )
{
  FILE *out;

  out = fopen( filename, "wb" );
  if( !out ) return OD_ERROR;
  fwrite( image, 1, csize, out );
```

```
  fclose( out );
  return OD_OK;
} /* odCameraImageToFile */


char *odCameraTakePicture( struct timeval *time, int shutter_speed, int gain )
{
  int width = OD_PIC_WIDTH, height = OD_PIC_HEIGHT, size, dev = -1;
  char *image, *device = VIDEO_DEV, *file = "test.ppm", *filejpeg = "test.jpg";
  char *fimage;
  int input = INPUT_DEFAULT; /* this means take over current device settings*/
  int norm  = NORM_DEFAULT;

//  FILE *out = stdout;
/*
  if (file) {
    out = fopen (filejpeg, "wb");
    if (!out) {
      printf( "error opening file\n" );
      perror (file);
      return 1;
    }
  }
*/

  /* open the video4linux device */
  dev = open( device, O_RDWR );
  if( dev == -1 ) {
    fprintf( stderr, "Error: Can't open device\n" );
    return OD_ERROR;
  }

  if( v4l_set_input( dev, input, norm ) == -1 ) return OD_ERROR;
  if( v4l_check_size( dev, &width, &height ) == -1 ) return OD_ERROR;


//  printf( "enter shutter speed: " );
//  scanf( "%d", &speed );
//  speed = 65535;
  if( ioctl( dev, VIDIOCPWCSSHUTTER, &shutter_speed ) == -1 ) return OD_ERROR;
  if( ioctl( dev, VIDIOCPWCSAGC, &gain ) == -1 ) return OD_ERROR;

  image = get_image( dev, width, height, &size );
  gettimeofday( time, NULL );

  if( !size ) {
    close( dev );
  }
  if( image ) {
    fimage = malloc( size );
    memcpy( fimage, image, size );
    //put_image_ppm (out, image, width, height );
    //put_image_jpeg(out, image, width, height, 60, VIDEO_PALETTE_YUV420P );

    if( size ) {
      munmap( image, size );
      close( dev );
    } else if( image ) {
      free( image );
    }
  } else {
    fprintf( stderr, "Error: Can't get image\n" );
    return NULL;
```

```
  }

  csize = size;

  return fimage;
} /* odTakePicture */


int odCameraMove( int pan, int tilt )
{
  int dev;
  struct pwc_mpt_angles values;
  int rval = OD_OK;

  if(( pan < -7000 ) || ( pan > 7000 )) return OD_ERROR;
  if(( tilt < -3000 ) || ( tilt > 2500 )) return OD_ERROR;

  dev = open( VIDEO_DEV, O_RDWR );
  if( dev == -1 ) {
    return OD_ERROR;
  }

  values.absolute = 1;
  if( ioctl(dev, VIDIOCPWCMPTGANGLE, &values) == -1 ) {
    fprintf( stderr, "Error: can't set pan / tilt\n" );
    rval = OD_ERROR;
  }
  values.pan = pan;
  values.tilt = tilt;
  if( ioctl(dev, VIDIOCPWCMPTSANGLE, &values) == -1 ) {
    fprintf( stderr, "Error: can't set pan / tilt\n" );
    rval = OD_ERROR;
  }

  close( dev );

  return rval;
} /* odCameraMove */


/* eof */


###### Subvehicle/refs/od/movecam.c #####

#include <stdlib.h>
#include <stdio.h>

#include "od_Camera.h"
#include "od_common.h"

int main( int argc, char **argv )
{
  int pan, tilt;

  if( argc != 3 ) return 0;

  pan = atoi( argv[1] );
  tilt = atoi( argv[2] );
  if( OD_OK == odCameraMove( pan, tilt )) {
    printf( "$GPCAM,%d,%d\n", pan, tilt );
  }
```

```
  return 0;
}

###### Subvehicle/refs/od/otherlibs/debug.h #####

#ifndef debugscripts

//Compile with debuging on
 //#define debugscript(x) printf(x)
 //#define debugcode(x) x

//Compile with debuging off
 #define debugscript(x)
 #define debugcode(x)

 #define attempt() {debugscript(__FUNCTION__ ": Attempt\n");}
 #define failure(x) {debugcode(printf(__FUNCTION__ ": Failure : %s",x));return 0;}
 #define success() {debugscript(__FUNCTION__ ": Success\n");return 1;}
 #define success_return(x) {debugscript(__FUNCTION__ ": Success\n");return x;}

 #define debugscripts

#endif


###### Subvehicle/refs/od/otherlibs/pwc-ioctl.h #####

#ifndef PWC_IOCTL_H
#define PWC_IOCTL_H

/* (C) 2001-2003 Nemosoft Unv.    webcam@smcc.demon.nl

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

/*         This is pwc-ioctl.h belonging to PWC 8.10                       */

/*
   Changes
   2001/08/03  Alvarado   Added ioctl constants to access methods for
                          changing white balance and red/blue gains
   2002/12/15  G. H. Fernandez-Toribio   VIDIOCGREALSIZE
 */

/* These are private ioctl() commands, specific for the Philips webcams.
   They contain functions not found in other webcams, and settings not
   specified in the Video4Linux API.

   The #define names are built up like follows:
   VIDIOC          VIDeo IOCtl prefix
         PWC               Philps WebCam
```

```
            G           optional: Get
            S           optional: Set
             ...   the function
 */



/* The frame rate is encoded in the video_window.flags parameter using
   the upper 16 bits, since some flags are defined nowadays. The following
   defines provide a mask and shift to filter out this value.

   In 'Snapshot' mode the camera freezes its automatic exposure and colour
   balance controls.
 */
#define PWC_FPS_SHIFT                16
#define PWC_FPS_MASK                 0x00FF0000
#define PWC_FPS_FRMASK               0x003F0000
#define PWC_FPS_SNAPSHOT    0x00400000



struct pwc_probe
{
         char name[32];
         int type;
};


/* pwc_whitebalance.mode values */
#define PWC_WB_INDOOR                0
#define PWC_WB_OUTDOOR               1
#define PWC_WB_FL           2
#define PWC_WB_MANUAL                3
#define PWC_WB_AUTO                  4

/* Used with VIDIOCPWC[SG]AWB (Auto White Balance).
   Set mode to one of the PWC_WB_* values above.
   *red and *blue are the respective gains of these colour components inside
   the camera; range 0..65535
   When 'mode' == PWC_WB_MANUAL, 'manual_red' and 'manual_blue' are set or read;
   otherwise undefined.
   'read_red' and 'read_blue' are read-only.
*/

struct pwc_whitebalance
{
         int mode;
         int manual_red, manual_blue;          /* R/W */
         int read_red, read_blue;    /* R/O */
};

/*
   'control_speed' and 'control_delay' are used in automatic whitebalance mode,
   and tell the camera how fast it should react to changes in lighting, and
   with how much delay. Valid values are 0..65535.
*/
struct pwc_wb_speed
{
         int control_speed;
         int control_delay;

};
```

```
/* Used with VIDIOCPWC[SG]LED */
struct pwc_leds
{
         int led_on;                          /* Led on-time; range = 0..25000 */
         int led_off;                         /* Led off-time; range = 0..25000  */
};

/* Image size (used with GREALSIZE) */
struct pwc_imagesize
{
         int width;
         int height;
};

/* Defines and structures for Motorized Pan & Tilt */
#define PWC_MPT_PAN                  0x01
#define PWC_MPT_TILT                 0x02
#define PWC_MPT_TIMEOUT              0x04 /* for status */

/* Set angles; when absolute = 1, the angle is absolute and the
   driver calculates the relative offset for you. This can only
   be used with VIDIOCPWCSANGLE; VIDIOCPWCGANGLE always returns
   absolute angles.
 */
struct pwc_mpt_angles
{
         int absolute;               /* write-only */
         int pan;           /* degrees * 100 */
         int tilt;          /* degress * 100 */
         int zoom;          /* N/A, set to -1 */
};

/* Range of angles of the camera, both horizontally and vertically.
   The zoom is not used, maybe in the future...

 */
struct pwc_mpt_range
{
         int pan_min, pan_max;                /* degrees * 100 */
         int tilt_min, tilt_max;
         int zoom_min, zoom_max;              /* -1, -1 */
};

struct pwc_mpt_status
{
         int status;
         int time_pan;
         int time_tilt;
};


 /* Restore user settings */
#define VIDIOCPWCRUSER               _IO('v', 192)
 /* Save user settings */
#define VIDIOCPWCSUSER               _IO('v', 193)
 /* Restore factory settings */
#define VIDIOCPWCFACTORY    _IO('v', 194)

 /* You can manipulate the compression factor. A compression preference of 0
    means use uncompressed modes when available; 1 is low compression, 2 is
    medium and 3 is high compression preferred. Of course, the higher the
    compression, the lower the bandwidth used but more chance of artefacts
```

```
    in the image. The driver automatically chooses a higher compression when
    the preferred mode is not available.
  */
 /* Set preferred compression quality (0 = uncompressed, 3 = highest compression) */
#define VIDIOCPWCSCQUAL              _IOW('v', 195, int)
 /* Get preferred compression quality */
#define VIDIOCPWCGCQUAL              _IOR('v', 195, int)


 /* This is a probe function; since so many devices are supported, it
    becomes difficult to include all the names in programs that want to
    check for the enhanced Philips stuff. So in stead, try this PROBE;
    it returns a structure with the original name, and the corresponding
    Philips type.
    To use, fill the structure with zeroes, call PROBE and if that succeeds,
    compare the name with that returned from VIDIOCGCAP; they should be the
    same. If so, you can be assured it is a Philips (OEM) cam and the type
    is valid.
 */
#define VIDIOCPWCPROBE               _IOR('v', 199, struct pwc_probe)

 /* Set AGC (Automatic Gain Control); int < 0 = auto, 0..65535 = fixed */
#define VIDIOCPWCSAGC                _IOW('v', 200, int)
 /* Get AGC; int < 0 = auto; >= 0 = fixed, range 0..65535 */
#define VIDIOCPWCGAGC                _IOR('v', 200, int)
 /* Set shutter speed; int < 0 = auto; >= 0 = fixed, range 0..65535 */
#define VIDIOCPWCSSHUTTER   _IOW('v', 201, int)

 /* Color compensation (Auto White Balance) */
#define VIDIOCPWCSAWB           _IOW('v', 202, struct pwc_whitebalance)
#define VIDIOCPWCGAWB           _IOR('v', 202, struct pwc_whitebalance)

 /* Auto WB speed */
#define VIDIOCPWCSAWBSPEED  _IOW('v', 203, struct pwc_wb_speed)
#define VIDIOCPWCGAWBSPEED  _IOR('v', 203, struct pwc_wb_speed)

 /* LEDs on/off/blink; int range 0..65535 */
#define VIDIOCPWCSLED           _IOW('v', 205, struct pwc_leds)
#define VIDIOCPWCGLED           _IOR('v', 205, struct pwc_leds)

  /* Contour (sharpness); int < 0 = auto, 0..65536 = fixed */
#define VIDIOCPWCSCONTOUR   _IOW('v', 206, int)
#define VIDIOCPWCGCONTOUR   _IOR('v', 206, int)

  /* Backlight compensation; 0 = off, otherwise on */
#define VIDIOCPWCSBACKLIGHT _IOW('v', 207, int)
#define VIDIOCPWCGBACKLIGHT _IOR('v', 207, int)

  /* Flickerless mode; = 0 off, otherwise on */
#define VIDIOCPWCSFLICKER   _IOW('v', 208, int)
#define VIDIOCPWCGFLICKER   _IOR('v', 208, int)

  /* Dynamic noise reduction; 0 off, 3 = high noise reduction */
#define VIDIOCPWCSDYNNOISE  _IOW('v', 209, int)
#define VIDIOCPWCGDYNNOISE  _IOR('v', 209, int)

 /* Real image size as used by the camera; tells you whether or not there's a gray
border around the image */
#define VIDIOCPWCGREALSIZE  _IOR('v', 210, struct pwc_imagesize)

 /* Motorized pan & tilt functions */
#define VIDIOCPWCMPTRESET   _IOW('v', 211, int)
#define VIDIOCPWCMPTGRANGE  _IOR('v', 211, struct pwc_mpt_range)
```

```
#define VIDIOCPWCMPTSANGLE  _IOW('v', 212, struct pwc_mpt_angles)
#define VIDIOCPWCMPTGANGLE  _IOR('v', 212, struct pwc_mpt_angles)
#define VIDIOCPWCMPTSTATUS  _IOR('v', 213, struct pwc_mpt_status)

#endif

###### Subvehicle/refs/od/otherlibs/v4l.h #####

/*
 * v4l.h
 *
 * Copyright (C) 2001 Rasca, Berlin
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#ifndef __V4L_H__
#define __V4L_H__

#define INPUT_DEFAULT       -1
#define NORM_DEFAULT        -1

#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif

int v4l_set_input (int fd, int input, int norm);
int v4l_check_size (int fd, int *width, int *height);
int v4l_check_palette (int fd, int *palette);
int v4l_mute_sound (int fd);
int v4l_check_mmap (int fd, int *size);
int v4l_yuv420p2rgb (unsigned char *, unsigned char *, int, int, int);
int v4l_yuv422p2rgb (unsigned char *, unsigned char *, int, int, int);
#endif

###### Subvehicle/refs/od/otherlibs/philipscamera.h #####

/*
 * philipscamera.c
 * This file is part of PhilipsWebCamLib
 *
 * PhilipsWebCamLib is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * PhilipsWebCamLib is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with PhilipsWebCamLib; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 *
 * PhilipsWebCamLib - Version 1.2.0
 * Release Date: 1/04
 * Author : Derrick Parkhurst
 * Copyright (c) 2004 Derrick Parkhurst <derrick.parkhurst@jhu.edu>
 * All Rights Reserved.
 *
 */

int Open_All_Devices();
int Open_Device (char *devname);
int Close_All_Devices ();
int Close_Device (int devhandle);
int Get_All_Device_Info ();
int Get_Capabilities (int devnum);
int Get_Picture (int devnum);
int Set_Picture(int devnum,
                int b,
                int h,
                int cl,
                int cn,
               int w);
int Get_Window (int devnum);
int Get_MMap (int devnum);
int Activate_MMap (int devnum);
int Deactivate_MMap (int devnum);
int MSync(int devnum, int buffer);
int MCapture(int devnum, int buffer);
int Set_Size(int devnum, int width, int height);
int Get_Frame_Rate(int devnum);
int Set_Frame_Rate(int devnum, int fps);
int Read_Image(int devnum, unsigned char *dst);
int Select_Input(int devnum, int number);
int Get_Width(int devnum);
int Get_Height(int devnum);
int Set_Led(int devnum, int on_time, int off_time);

void Dump_Current_Settings(int devnum);
void Query_Pan_Tilt_Status(int devnum);
void Reset_Pan_Tilt(int devnum, int what); //reset pan(bit 0) and/or tilt(bit 1)
void Set_Pan_Or_Tilt_Or_Zoom(int devnum, char what, int value);
void Set_Framerate(int devnum, int framerate); // (0...63)
void Flash_Settings(int devnum);
void Restore_Settings(int devnum);
void Restore_Factory_Settings(int devnum);
void Set_Compression_Preference(int devnum, int pref); // (0..3)
void Set_Automatic_Gain_Control(int devnum, int pref); // (0...65535)
void Set_Shutter_Speed(int devnum, int pref); // (1...65535)
void Set_Automatic_White_Balance_Mode(int devnum, char *mode);
//(auto/manual/indoor/outdoor/fl)
void Set_Automatic_White_Balance_Mode_Red(int devnum, int val); //(0...65535)
void Set_Automatic_White_Balance_Mode_Blue(int devnum, int val); //(0..65536)
void Set_Automatic_White_Balance_Speed(int devnum, int val); //(1..65536)
void Set_Automatic_White_Balance_Delay(int devnum, int val); //(1..65536)
void Set_LED_On_Time(int devnum, int val); //(0...25500ms)
void Set_LED_Off_Time(int devnum, int val);
```

```
void Set_Sharpness(int devnum, int val); //(0...65535)
void Set_Backlight_Compensation(int devnum, int val); //(0=0ff, other=on)
void Set_Antiflicker_Mode(int devnum, int val); //(0=0ff, other=on)
void Set_Noise_Reduction(int devnum, int val); //(0=none...3=high)
void Set_PanTilt(int devnum, int pan, int tilt);
void Get_PanTilt(int devnum, int *pan, int *tilt);


###### Subvehicle/refs/od/otherlibs/moveit.c #####

/* moveit.c
 *
 * example file on how to move the camera
 *
 */

#include "philipscamera.h"


int main( )
{
  int dev;
  int val, val2;

  dev = Open_All_Devices( );
  Get_All_Device_Info( );
/*
  Set_Pan_Or_Tilt_Or_Zoom( dev, 1, 0 );
  sleep( 2 );
  Set_Pan_Or_Tilt_Or_Zoom( dev, 0, 0 );
*/
  Set_PanTilt( dev, 0, 0 );
  printf( "how much to move pan? " );
  scanf( "%d", &val );
  printf( "how much do move tilt? " );
  scanf( "%d", &val2 );
  Set_PanTilt( dev, val, val2 );
/*
  Set_Pan_Or_Tilt_Or_Zoom( dev, 0, val );
  sleep( 2 );
  Set_Pan_Or_Tilt_Or_Zoom( dev, 1, val );
*/

  Close_All_Devices( );
  return 0;
}

###### Subvehicle/refs/od/otherlibs/philipscameraARM.c #####

/*
 * philipscamera.c
 * This file is part of PhilipsWebCamLib
 *
 * PhilipsWebCamLib is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * PhilipsWebCamLib is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
```

```
 * You should have received a copy of the GNU General Public License
 * along with PhilipsWebCamLib; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 *
 * PhilipsWebCamLib - Version 1.2.0
 * Release Date: 1/04
 * Author : Derrick Parkhurst
 * Copyright (c) 2004 Derrick Parkhurst <derrick.parkhurst@jhu.edu>
 * All Rights Reserved.
 *
 *
 * This code was derived in part from the GNU GPL project
 * setpwc (C) 2003 Folkert van Heusden folkert@vanheusden.com
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
#include <linux/videodev.h>
#include <sys/ioctl.h>
#include <string.h>
#include <sys/mman.h>
//#include "ccvt.h"
#include "pwc-ioctl.h"
#include "debug.h"

#define MAXDEVICES 8
#define SET_PAN             0
#define SET_TILT   1
#define SET_ZOOM   2

int devcount=0;
int device[MAXDEVICES];
struct video_capability devicecapabilities[MAXDEVICES];
struct video_picture devicepictures[MAXDEVICES];
struct video_window devicewindows[MAXDEVICES];
struct video_mbuf devicebuffers[MAXDEVICES];
unsigned char *videobuffers[MAXDEVICES];


int Get_Width(int devnum) { return devicewindows[devnum].width; }
int Get_Height(int devnum) { return devicewindows[devnum].height; }

int Open_All_Devices() {
   char devname[40];
   struct stat devstat;
   int i;

   attempt();

   for (i = 0; i < MAXDEVICES; i++) {
     sprintf(devname, "/dev/video%d", i);
     if (stat(devname, &devstat) == 0) {
       if (!S_ISLNK(devstat.st_mode)) {
         device[devcount]=Open_Device(devname);
```

```
         if (device[devcount]>=0) {
           printf("Device num : %d, fd : %d, name : %s successfully opened\n",
devcount, device[devcount], devname);
             devcount++;
         }
       }
     }
   }
   if (devcount>0) {success_return(0);} else {failure("No devices opened\n");}
}

int Open_Device (char *devname) {
  int dev;

  return open(devname, O_RDONLY);
}

int Close_All_Devices () {

 int i;

   attempt();

   for (i = 0; i < devcount; i++) {
     if (Close_Device(device[i])==0) {
       printf("Device successfully closed\n");
     } else {
       failure("Error: Device that was opened was not closed\n");
     }
   }

   success();
}

int Close_Device (int devhandle) {

  return close(devhandle);

}

int Get_All_Device_Info () {

 int i;

  attempt();

   for (i = 0; i < devcount; i++) {
     Get_Capabilities (i);
     Get_Picture (i);
     Get_Window (i);
     Get_MMap (i);
   }

  success();
}

int Get_Capabilities (int devnum) {

 attempt();

 if (ioctl(device[devnum], VIDIOCGCAP, &devicecapabilities[devnum]) < 0) {
  failure("IOCTL Error\n");
 } else {
```

```
  printf("**Device Capabilites**\n");
  printf("Name:%s\n",devicecapabilities[devnum].name);
  printf("Type:%d\n",devicecapabilities[devnum].type);
  printf("Channels:%d\n",devicecapabilities[devnum].channels);
  printf("Audio Channels:%d\n",devicecapabilities[devnum].audios);
  printf("Max Width:%d\n",devicecapabilities[devnum].maxwidth);
  printf("Min Width:%d\n",devicecapabilities[devnum].minwidth);
  printf("Max Height:%d\n",devicecapabilities[devnum].maxheight);
  printf("Min Height:%d\n",devicecapabilities[devnum].minheight);
  success();
 }
}

int Get_Picture (int devnum) {

 attempt();

 if (ioctl(device[devnum], VIDIOCGPICT, &devicepictures[devnum]) < 0) {
  failure("IOCTL Error\n");
 } else {
  printf("**Device Picture**\n");
  printf("Brightness:%d\n",devicepictures[devnum].brightness);
  printf("Hue:%d\n",devicepictures[devnum].hue);
  printf("Colour:%d\n",devicepictures[devnum].colour);
  printf("Contrast:%d\n",devicepictures[devnum].contrast);
  printf("Whiteness:%d\n",devicepictures[devnum].whiteness);
  printf("Depth:%d\n",devicepictures[devnum].depth);
  printf("Palette:%d\n",devicepictures[devnum].palette);
  if (devicepictures[devnum].palette==VIDEO_PALETTE_YUV420P) {
   printf("Palette: YUV 4:2:0 Planar\n");
  }
  success();
 }
}

int Set_Picture(int devnum,
                int b,
                int h,
                int cl,
                int cn,
               int w)
{
   attempt();

   if (device[devnum] < 0) failure("Invalid device");

   if (b>=0) devicepictures[devnum].brightness = b & 0xffff;
   if (h>=0) devicepictures[devnum].hue = h & 0xffff;
   if (cl>=0) devicepictures[devnum].colour = cl & 0xffff;
   if (cn>=0) devicepictures[devnum].contrast = cn & 0xffff;
   if (w>=0) devicepictures[devnum].whiteness = w & 0xffff;

   if (ioctl(device[devnum], VIDIOCSPICT, &devicepictures[devnum]) < 0) failure("IOCTL
Error\n");
   success();
}



int Get_Window (int devnum) {

 attempt();
```

```
 if (ioctl(device[devnum], VIDIOCGWIN, &devicewindows[devnum]) < 0) {
  failure("IOCTL Error\n");
 } else {
  printf("**Device Window**\n");
  printf("x,y:(%d,%d)\n",devicewindows[devnum].x,devicewindows[devnum].y);
  printf("Width,Height:(%d,%d)\n",devicewindows[devnum].width,
                                        devicewindows[devnum].height);
  success_return(0);
 }
}

int Get_MMap (int devnum) {
 int i;

 attempt();

   if (ioctl(device[devnum], VIDIOCGMBUF, &devicebuffers[devnum]) < 0) {
    failure("IOCTL Error\n");
   } else {
    printf("**Device Memory**\n");
    printf("Size:%d\n",devicebuffers[devnum].size);
    printf("Frames:%d\n",devicebuffers[devnum].frames);
    for (i=0; i<devicebuffers[devnum].frames;i++) {
     printf("Frame[%d]:%p\n",i,devicebuffers[devnum].offsets[i]);
    }
    success();
   }
}


int Activate_MMap (int devnum) {
 char err[255];

 attempt();

 if (devicebuffers[devnum].size > 0) {
  videobuffers[devnum] = (unsigned char *)mmap(NULL,devicebuffers[devnum].size,
PROT_READ, MAP_SHARED, device[devnum], 0);
  debugcode(printf("mmap() Memory Buffer : (%p)\n", videobuffers[devnum]));
  if (videobuffers[devnum] == (unsigned char *)-1) {
   sprintf(err,"mmap() failed (%d)\n", errno);
   devicebuffers[devnum].size = 0;
   failure(err);
  }
 }

 success();
}

int Deactivate_MMap (int devnum) {

 attempt();

 if (devicebuffers[devnum].size > 0) {
  munmap(videobuffers[devnum], devicebuffers[devnum].size);
  devicebuffers[devnum].size = 0;
  videobuffers[devnum] = NULL;
 }

 success();
}

int MSync(int devnum, int buffer) {
```

```
 char err[255];

 attempt();

   if (devicebuffers[devnum].size == 0) failure("Device Buffer Size<=0\n");

   if (ioctl(device[devnum], VIDIOCSYNC, &buffer) < 0) {
    sprintf(err,"MSync: Error %d\n",errno);
    failure(err);
   }

   success();
}

int MCapture(int devnum, int buffer)
{
   struct video_mmap vm;
   char err[255];

   attempt();

   if (devicebuffers[devnum].size == 0) failure("Error: Device Buffer Size<=0\n");

   vm.frame = buffer;
   vm.format = devicepictures[devnum].palette;
   vm.width = devicewindows[devnum].width;
   vm.height = devicewindows[devnum].height;
   if (ioctl(device[devnum], VIDIOCMCAPTURE, &vm) < 0) {
    sprintf(err,"MCapture Error:%d\n",errno);
    failure(err);
   }

   success();
}

int Set_Size(int devnum, int width, int height)
{
   struct video_window vwin;

   attempt();

   if (device[devnum] < 0 ||
       width > devicecapabilities[devnum].maxwidth ||
       height > devicecapabilities[devnum].maxheight ||
       width < devicecapabilities[devnum].minwidth ||
       height < devicecapabilities[devnum].minheight) failure("Parameter or device out
of range\n");

   if (ioctl(device[devnum], VIDIOCGWIN, &vwin) < 0) failure("IOCTL error\n");


   vwin.width = width;
   vwin.height = height;
   vwin.clipcount = 0;
   if (ioctl(device[devnum], VIDIOCSWIN, &vwin) < 0) failure("IOCTL error\n");

   Get_Window (devnum);


   success();
}

int Get_Frame_Rate(int devnum) {
```

```
   struct video_window vwin;

   attempt();

   if (device[devnum] < 0) failure("Invalid device\n");
   if (ioctl(device[devnum], VIDIOCGWIN, &vwin) < 0) failure("IOCTL error\n");

   success_return (((vwin.flags & PWC_FPS_FRMASK) >> PWC_FPS_SHIFT));
}

int Set_Frame_Rate(int devnum, int fps)
{
   struct video_window vwin;

   attempt();

   if (device[devnum] < 0) return(-1);
   if (ioctl(device[devnum], VIDIOCGWIN, &vwin) < 0) failure("IOCTL error\n");
   vwin.flags = (vwin.flags & ~PWC_FPS_MASK) | (fps << PWC_FPS_SHIFT);
   if (ioctl(device[devnum], VIDIOCSWIN, &vwin) < 0) failure("IOCTL error\n");

   success();

}


int Read_Image(int devnum, unsigned char *dst)
{
   unsigned char *src;
   int i, n;

   attempt();

   if (devicebuffers[devnum].size <= 0) failure("Error: buffersize<=0\n");

   printf( "MCapture starting\n" );
   MCapture(devnum,0);
   printf( "MSync starting\n" );
   MSync(devnum,0);
   printf( "videobuffers = %x\n", (int)videobuffers );
   printf( "devicebuffers = %x\n", (int)devicebuffers );
//   printf( "other1 = %x", videobuffers[devnum] );
   printf( "other = %x", devicebuffers[devnum].offsets[0] );
   src = videobuffers[devnum] + devicebuffers[devnum].offsets[0];
//   n = devicewindows[devnum].width * devicewindows[devnum].height;

// HARD CODED IN, THIS SHOULD NOT BE LIKE THIS
   printf( "%d", (int)src );
   memcpy( dst, src, 640 * 480 * 4 );

   success();

}



int Save_Image(char *filename, unsigned char *data, int x, int y)
{
 int            channels,datasize;
 size_t            channelsize;
 FILE              *outfile;

 attempt();
```

```
 if (data==NULL) failure("Unallocated Data\n");
 if (!(outfile=fopen(filename,"w+b"))) failure("Couldn't create output file\n");
 channels=1;
 datasize=1;
 channelsize=x*y;
 fwrite(&x,2,1,outfile);
 fwrite(&y,2,1,outfile);
 fwrite(&channels,2,1,outfile);
 fwrite(&datasize,2,1,outfile);
 fwrite(data,1,channelsize,outfile);
 fclose(outfile);

 success();
}

int Select_Input(int devnum, int number)
{
   struct video_channel arg;
   char err[255];

   attempt();

   if (device[devnum] < 0) failure("Error: Bad Device Num\n");

   arg.channel = number;

   if (ioctl(device[devnum], VIDIOCGCHAN, &arg) < 0) {
    sprintf(err,"Select Input Error: %d\n",errno);
    failure(err);
   }

   if (ioctl(device[devnum], VIDIOCSCHAN, &arg) < 0) {
    sprintf(err,"Select Input Error: %d\n",errno);
    failure(err);
   }

   success();
}


int Set_Led(int devnum, int on_time, int off_time)
{
   struct pwc_leds leds;

   attempt();

   if (device[devnum] < 0) failure("Error: Bad Device Num\n");
   leds.led_on = on_time;
   leds.led_off = off_time;
   if (ioctl(device[devnum], VIDIOCPWCSLED, &leds) < 0) return(-1);

   success();

}

void error_exit(char *what_ioctl)
{
         fprintf(stderr, "Error while doing ioctl %s: %s\n", what_ioctl,
strerror(errno));

         /* commented out: some versions of the driver seem to return
          * unexpected errors */
```

```
        /* exit(1); */
}

void not_supported(char *what)
{
        printf("%s is not supported by the combination\n", what);
        printf("of your webcam and the driver.\n");
}

void Dump_Current_Settings(int devnum)
{
        struct video_capability vcap;
        struct video_window vwin;
        struct video_picture vpic;
        struct pwc_probe pwcp;
        int dummy;
        struct pwc_whitebalance pwcwb;
        struct pwc_leds pwcl;
        struct pwc_mpt_range pmr;
        struct pwc_mpt_angles pma;

        /* get name */
        if (ioctl(device[devnum], VIDIOCGCAP, &vcap) == -1)
                error_exit("VIDIOCGCAP");
        printf("Current device: %s\n", vcap.name);

        /* verify that it IS a Philips Webcam */
        if (ioctl(device[devnum], VIDIOCPWCPROBE, &pwcp) == -1)
                error_exit("VIDIOCPWCPROBE");
        if (strcmp(vcap.name, pwcp.name) != 0)
                printf("Warning: this might not be a Philips compatible
webcam!\n");
        printf("VIDIOCPWCPROBE returns: %s - %d\n", pwcp.name, pwcp.type);

        /* get resolution/framerate */
        if (ioctl(device[devnum], VIDIOCGWIN, &vwin) == -1)
                error_exit("VIDIOCGWIN");
        printf("Resolution (x, y): %d, %d\n", vwin.width, vwin.height);
        printf("Offset: %d, %d\n", vwin.x, vwin.y);
        if (vwin.flags & PWC_FPS_FRMASK)
                printf("Framerate: %d\n", (vwin.flags & PWC_FPS_FRMASK) >>
PWC_FPS_SHIFT);

        /* color (etc.) settings */
        if (ioctl(device[devnum], VIDIOCGPICT, &vpic) == -1)
                error_exit("VIDIOCGPICT");
        printf("Brightness: %d\n", vpic.brightness);
        printf("Hue: %d\n", vpic.hue);
        printf("Colour: %d\n", vpic.colour);
        printf("Contrast: %d\n", vpic.contrast);
        printf("Whiteness: %d\n", vpic.whiteness);
        printf("Palette: ");
        switch(vpic.palette) {
        case VIDEO_PALETTE_GREY:
                printf("Linear intensity grey scale (255 is brightest).\n");
                break;
        case VIDEO_PALETTE_HI240:
                printf("The BT848 8bit colour cube.\n");
                break;
        case VIDEO_PALETTE_RGB565:
                printf("RGB565 packed into 16 bit words.\n");
                break;
        case VIDEO_PALETTE_RGB555:
```

```
                printf("RGV555 packed into 16 bit words, top bit undefined.\n");
                break;
        case VIDEO_PALETTE_RGB24:
                printf("RGB888 packed into 24bit words.\n");
                break;
        case VIDEO_PALETTE_RGB32:
                printf("RGB888 packed into the low 3 bytes of 32bit words. The top
8bits are undefined.\n");
                break;
        case VIDEO_PALETTE_YUV422:
                printf("Video style YUV422 - 8bits packed 4bits Y 2bits U 2bits
V\n");
                break;
        case VIDEO_PALETTE_YUYV:
                printf("Describe me\n");
                break;
        case VIDEO_PALETTE_UYVY:
                printf("Describe me\n");
                break;
        case VIDEO_PALETTE_YUV420:
                printf("YUV420 capture\n");
                break;
        case VIDEO_PALETTE_YUV411:
                printf("YUV411 capture\n");
                break;
        case VIDEO_PALETTE_RAW:
                printf("RAW capture (BT848)\n");
                break;
        case VIDEO_PALETTE_YUV422P:
                printf("YUV 4:2:2 Planar\n");
                break;
        case VIDEO_PALETTE_YUV411P:
                printf("YUV 4:1:1 Planar\n");
                break;
        case VIDEO_PALETTE_YUV420P:
                printf("YUV 4:2:0 Planar\n");
                break;
        case VIDEO_PALETTE_YUV410P:
                printf("YUV 4:1:0 Planar\n");
                break;
        default:
                printf("Unknown! (%d)\n", vpic.palette);
        }

        if (ioctl(device[devnum], VIDIOCPWCGCQUAL, &dummy) == -1)
                error_exit("VIDIOCPWCGCQUAL");
        printf("Compression preference: %d\n", dummy);

        if (ioctl(device[devnum], VIDIOCPWCGAGC, &dummy) == -1)
                error_exit("VIDIOCPWCGAGC");
        printf("Automatic gain control: %d\n", dummy);

        if (ioctl(device[devnum], VIDIOCPWCGAWB, &pwcwb) == -1)
                error_exit("VIDIOCPWCGAWB");
        printf("Whitebalance mode: ");
        if (pwcwb.mode == PWC_WB_AUTO)
                printf("auto\n");
        else if (pwcwb.mode == PWC_WB_MANUAL)
                printf("manual (red: %d, blue: %d)\n", pwcwb.manual_red,
pwcwb.manual_blue);
        else if (pwcwb.mode == PWC_WB_INDOOR)
                printf("indoor\n");
        else if (pwcwb.mode == PWC_WB_OUTDOOR)
```

```
        printf("outdoor\n");
else if (pwcwb.mode == PWC_WB_FL)
        printf("artificial lightning ('fl')\n");
else
        printf("unknown!\n");

if (ioctl(device[devnum], VIDIOCPWCGLED, &pwcl) != -1)
{
        printf("Led ON time: %d\n", pwcl.led_on);
        printf("Led OFF time: %d\n", pwcl.led_off);
}
else
{
        not_supported("Blinking of LED");
}

if (ioctl(device[devnum], VIDIOCPWCGCONTOUR, &dummy) == -1)
        error_exit("VIDIOCPWCGCONTOUR");
printf("Sharpness: %d\n", dummy);

if (ioctl(device[devnum], VIDIOCPWCGBACKLIGHT, &dummy) == -1)
        error_exit("VIDIOCPWCGBACKLIGHT");
printf("Backlight compensation mode: ");
if (dummy == 0) printf("off\n"); else printf("on\n");

if (ioctl(device[devnum], VIDIOCPWCGFLICKER, &dummy) != -1)
{
        printf("Anti-flicker mode: ");
        if (dummy == 0) printf("off\n"); else printf("on\n");
}
else
{
        not_supported("Anti-flicker mode");
}

if (ioctl(device[devnum], VIDIOCPWCGDYNNOISE, &dummy) != -1)
{
        printf("Noise reduction mode: %d ", dummy);
        if (dummy == 0) printf("(none)");
        else if (dummy == 3) printf("(high)");
        printf("\n");
}
else
{
        not_supported("Noise reduction mode");
}

if (ioctl(device[devnum], VIDIOCPWCMPTGRANGE, &pmr) == -1)
{
        not_supported("Pan/tilt range");
}
else
{
        printf("Pan min. : %d, max.: %d\n", pmr.pan_min, pmr.pan_max);
        printf("Tilt min.: %d, max.: %d\n", pmr.tilt_min, pmr.tilt_max);
        printf("Zoom min.: %d, max.: %d\n", pmr.zoom_min, pmr.zoom_max);
}

pma.absolute=1;
if (ioctl(device[devnum], VIDIOCPWCMPTGANGLE, &pma) == -1)
{
        not_supported("Get pan/tilt position");
}
```

```
        else
        {
                printf("Pan  (degrees * 100): %d\n", pma.pan);
                printf("Tilt (degrees * 100): %d\n", pma.tilt);
                printf("Zoom: %d\n", pma.zoom);
        }
}

void Query_Pan_Tilt_Status(int devnum)
{
        struct pwc_mpt_status pms;

        if (ioctl(device[devnum], VIDIOCPWCMPTSTATUS, &pms) == -1)
                error_exit("VIDIOCPWCMPTSTATUS");

        printf("Status: %d\n", pms.status);
        printf("Time pan: %d\n", pms.time_pan);
        printf("Time tilt: %d\n", pms.time_tilt);
}

void Reset_Pan_Tilt(int devnum, int what)
{
        if (ioctl(device[devnum], VIDIOCPWCMPTRESET, what) == -1)
                error_exit("VIDIOCPWCMPTRESET");
}

void Set_Pan_Or_Tilt_Or_Zoom(int devnum, char what, int value)
{
        struct pwc_mpt_angles pma;

        pma.absolute=1;
        if (ioctl(device[devnum], VIDIOCPWCMPTGANGLE, &pma) == -1)
                error_exit("VIDIOCPWCMPTGANGLE");

        if (what == SET_PAN)
                pma.pan = value;
        else if (what == SET_TILT)
                pma.tilt = value;
        else if (what == SET_ZOOM)
                pma.zoom = value;

        if (ioctl(device[devnum], VIDIOCPWCMPTSANGLE, &pma) == -1)
                error_exit("VIDIOCPWCMPTSANGLE");
}

void Set_PanTilt(int devnum, int pan, int tilt)
{
        struct pwc_mpt_angles pma;

        pma.absolute=1;
        if (ioctl(device[devnum], VIDIOCPWCMPTGANGLE, &pma) == -1)
                error_exit("VIDIOCPWCMPTGANGLE");

        pma.pan = pan;
        pma.tilt = tilt;

        if (ioctl(device[devnum], VIDIOCPWCMPTSANGLE, &pma) == -1)
                error_exit("VIDIOCPWCMPTSANGLE");
}

void Get_PanTilt(int devnum, int *pan, int *tilt)
{
        struct pwc_mpt_status pms;
```

```
	if (ioctl(device[devnum], VIDIOCPWCMPTSTATUS, &pms) == -1)
		error_exit("VIDIOCPWCMPTSTATUS");

	*pan=pms.time_pan;
	*tilt=pms.time_tilt;
}


void Set_Framerate(int devnum, int framerate)
{
	struct video_window vwin;

	/* get resolution/framerate */
	if (ioctl(device[devnum], VIDIOCGWIN, &vwin) == -1)
		error_exit("VIDIOCGWIN");

	if (vwin.flags & PWC_FPS_FRMASK)
	{
		/* set new framerate */
		vwin.flags &= ~PWC_FPS_FRMASK;
		vwin.flags |= (framerate << PWC_FPS_SHIFT);

		if (ioctl(device[devnum], VIDIOCSWIN, &vwin) == -1)
			error_exit("VIDIOCSWIN");
	}
	else
	{
		fprintf(stderr, "This device doesn't support setting the
framerate.\n");
		exit(1);
	}
}

void Flash_Settings(int devnum)
{
	if (ioctl(device[devnum], VIDIOCPWCSUSER) == -1)
		error_exit("VIDIOCPWCSUSER");
}

void Restore_Settings(int devnum)
{
	if (ioctl(device[devnum], VIDIOCPWCRUSER) == -1)
		error_exit("VIDIOCPWCRUSER");
}

void Restore_Factory_Settings(int devnum)
{
	if (ioctl(device[devnum], VIDIOCPWCFACTORY) == -1)
		error_exit("VIDIOCPWCFACTORY");
}

void Set_Compression_Preference(int devnum, int pref)
{
	if (ioctl(device[devnum], VIDIOCPWCSCQUAL, &pref) == -1)
		error_exit("VIDIOCPWCSCQUAL");
}

void Set_Automatic_Gain_Control(int devnum, int pref)
{
	if (ioctl(device[devnum], VIDIOCPWCSAGC, &pref) == -1)
		error_exit("VIDIOCPWCSAGC");
}
```

```
void Set_Shutter_Speed(int devnum, int pref)
{
	if (ioctl(device[devnum], VIDIOCPWCSSHUTTER, &pref) == -1)
		error_exit("VIDIOCPWCSSHUTTER");
}

void Set_Automatic_White_Balance_Mode(int devnum, char *mode)
{
	struct pwc_whitebalance pwcwb;

	if (ioctl(device[devnum], VIDIOCPWCGAWB, &pwcwb) == -1)
		error_exit("VIDIOCPWCGAWB");

	if (strcasecmp(mode, "auto") == 0)
		pwcwb.mode = PWC_WB_AUTO;
	else if (strcasecmp(mode, "manual") == 0)
		pwcwb.mode = PWC_WB_MANUAL;
	else if (strcasecmp(mode, "indoor") == 0)
		pwcwb.mode = PWC_WB_INDOOR;
	else if (strcasecmp(mode, "outdoor") == 0)
		pwcwb.mode = PWC_WB_OUTDOOR;
	else if (strcasecmp(mode, "fl") == 0)
		pwcwb.mode = PWC_WB_FL;
	else
	{
		fprintf(stderr, "'%s' is not a known white balance mode.\n", mode);
		exit(1);
	}

	if (ioctl(device[devnum], VIDIOCPWCSAWB, &pwcwb) == -1)
		error_exit("VIDIOCPWCSAWB");
}

void Set_Automatic_White_Balance_Mode_Red(int devnum, int val)
{
	struct pwc_whitebalance pwcwb;

	if (ioctl(device[devnum], VIDIOCPWCGAWB, &pwcwb) == -1)
		error_exit("VIDIOCPWCGAWB");

	pwcwb.manual_red = val;

	if (ioctl(device[devnum], VIDIOCPWCSAWB, &pwcwb) == -1)
		error_exit("VIDIOCPWCSAWB");
}

void Set_Automatic_White_Balance_Mode_Blue(int devnum, int val)
{
	struct pwc_whitebalance pwcwb;

	if (ioctl(device[devnum], VIDIOCPWCGAWB, &pwcwb) == -1)
		error_exit("VIDIOCPWCGAWB");

	pwcwb.manual_blue = val;

	if (ioctl(device[devnum], VIDIOCPWCSAWB, &pwcwb) == -1)
		error_exit("VIDIOCPWCSAWB");
}

void Set_Automatic_White_Balance_Speed(int devnum, int val)
{
	struct pwc_wb_speed pwcwbs;
```

```
        if (ioctl(device[devnum], VIDIOCPWCGAWBSPEED, &pwcwbs) == -1)
                error_exit("VIDIOCPWCGAWBSPEED");

        pwcwbs.control_speed = val;

        if (ioctl(device[devnum], VIDIOCPWCSAWBSPEED, &pwcwbs) == -1)
                error_exit("VIDIOCPWCSAWBSPEED");
}

void Set_Automatic_White_Balance_Delay(int devnum, int val)
{
        struct pwc_wb_speed pwcwbs;

        if (ioctl(device[devnum], VIDIOCPWCGAWBSPEED, &pwcwbs) == -1)
                error_exit("VIDIOCPWCGAWBSPEED");

        pwcwbs.control_delay = val;

        if (ioctl(device[devnum], VIDIOCPWCSAWBSPEED, &pwcwbs) == -1)
                error_exit("VIDIOCPWCSAWBSPEED");
}

void Set_LED_On_Time(int devnum, int val)
{
        struct pwc_leds pwcl;

        if (ioctl(device[devnum], VIDIOCPWCGLED, &pwcl) == -1)
                error_exit("VIDIOCPWCGLED");

        pwcl.led_on = val;

        if (ioctl(device[devnum], VIDIOCPWCSLED, &pwcl) == -1)
                error_exit("VIDIOCPWCSLED");
}

void Set_LED_Off_Time(int devnum, int val)
{
        struct pwc_leds pwcl;

        if (ioctl(device[devnum], VIDIOCPWCGLED, &pwcl) == -1)
                error_exit("VIDIOCPWCGLED");

        pwcl.led_off = val;

        if (ioctl(device[devnum], VIDIOCPWCSLED, &pwcl) == -1)
                error_exit("VIDIOCPWCSLED");
}

void Set_Sharpness(int devnum, int val)
{
        if (ioctl(device[devnum], VIDIOCPWCSCONTOUR, &val) == -1)
                error_exit("VIDIOCPWCSCONTOUR");
}

void Set_Backlight_Compensation(int devnum, int val)
{
        if (ioctl(device[devnum], VIDIOCPWCSBACKLIGHT, &val) == -1)
                error_exit("VIDIOCPWCSBACKLIGHT");
}

void Set_Antiflicker_Mode(int devnum, int val)
{
```

```
          if (ioctl(device[devnum], VIDIOCPWCSFLICKER, &val) == -1)
                    error_exit("VIDIOCPWCSFLICKER");
}

void Set_Noise_Reduction(int devnum, int val)
{
          if (ioctl(device[devnum], VIDIOCPWCSDYNNOISE, &val) == -1)
                    error_exit("VIDIOCPWCSDYNNOISE");
}

###### Subvehicle/refs/od/otherlibs/v4l.c #####

/*
 * v4l.c
 *
 * Copyright (C) 2001 Rasca, Berlin
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/types.h>
#include <linux/videodev.h>
#include "v4l.h"

#define min(a,b) ((a) < (b) ? (a) : (b))
#define max(a,b) ((a) > (b) ? (a) : (b))

/*
 * set the input and norm for the video4linux device
 */
int
v4l_set_input (int fd, int input, int norm)
{
          struct video_channel vid_chnl;

          if (input != INPUT_DEFAULT || norm != NORM_DEFAULT) {
                    if (vid_chnl.channel != INPUT_DEFAULT)
                              vid_chnl.channel = input;
                    else
                              vid_chnl.channel = 0;
                    vid_chnl.norm = -1;
                    if (ioctl (fd, VIDIOCGCHAN, &vid_chnl) == -1) {
                              perror ("ioctl (VIDIOCGCHAN)");
                              return -1;
```

```
                } else {
                        if (input != 0)
                                vid_chnl.channel = input;
                        if (norm != NORM_DEFAULT)
                                vid_chnl.norm    = norm;
                        if (ioctl (fd, VIDIOCSCHAN, &vid_chnl) == -1) {
                                perror ("ioctl (VIDIOCSCHAN)");
                                return -1;
                        }
                }
        }
        return 0;
}

/*
 * check the size and readjust if necessary
 */
int
v4l_check_size (int fd, int *width, int *height)
{
        struct video_capability vid_caps;

        if (ioctl (fd, VIDIOCGCAP, &vid_caps) == -1) {
                perror ("ioctl (VIDIOCGCAP)");
                return -1;
        }
        /* readjust if necessary */
        if (*width > vid_caps.maxwidth || *width < vid_caps.minwidth) {
                *width = min (*width, vid_caps.maxwidth);
                *width = max (*width, vid_caps.minwidth);
                fprintf (stderr, "readjusting width to %d\n", *width);
        }
        if (*height > vid_caps.maxheight || *height < vid_caps.minheight) {
                *height = min (*height, vid_caps.maxheight);
                *height = max (*height, vid_caps.minheight);
                fprintf (stderr, "readjusting height to %d\n", *height);
        }
        return 0;
}

/*
 * check the requested palette and adjust if possible
 * seems not to work :-(
 */
int
v4l_check_palette (int fd, int *palette)
{
        struct video_picture vid_pic;

        if (!palette)
                return -1;

        if (ioctl (fd, VIDIOCGPICT, &vid_pic) == -1) {
                perror ("ioctl (VIDIOCGPICT)");
                return -1;
        }
        vid_pic.palette = *palette;
        if (ioctl (fd, VIDIOCSPICT, &vid_pic) == -1) {
                /* try YUV420P
                 */
                fprintf (stderr, "failed\n");
                vid_pic.palette = *palette = VIDEO_PALETTE_YUV420P;
                if (ioctl (fd, VIDIOCSPICT, &vid_pic) == -1) {
```

```
                                perror ("ioctl (VIDIOCSPICT) to YUV");
                                /* ok, try grayscale..
                                 */
                                vid_pic.palette = *palette = VIDEO_PALETTE_GREY;
                                if (ioctl (fd, VIDIOCSPICT, &vid_pic) == -1) {
                                        perror ("ioctl (VIDIOCSPICT) to GREY");
                                        return -1;
                                }
                        }
                }
                return 0;
}

/*
 * check if driver supports mmap'ed buffer
 */
int
v4l_check_mmap (int fd, int *size)
{
                struct video_mbuf vid_buf;

                if (ioctl (fd, VIDIOCGMBUF, &vid_buf) == -1) {
                        return -1;
                }
                if (size)
                        *size = vid_buf.size;
                return 0;
}

/*
 * mute sound if available
 */
int
v4l_mute_sound (int fd)
{
                struct video_capability vid_caps;
                struct video_audio vid_aud;

                if (ioctl (fd, VIDIOCGCAP, &vid_caps) == -1) {
                        perror ("ioctl (VIDIOCGCAP)");
                        return -1;
                }
                if (vid_caps.audios > 0) {
                        /* mute the sound */
                        if (ioctl (fd, VIDIOCGAUDIO, &vid_aud) == -1) {
                                return -1;
                } else {
                    vid_aud.flags = VIDEO_AUDIO_MUTE;
                    if (ioctl (fd, VIDIOCSAUDIO, &vid_aud) == -1)
                                        return -1;
                }
            }
                return 0;
}

/*
 * Turn a YUV4:2:0 block into an RGB block
 *
 * Video4Linux seems to use the blue, green, red channel
 * order convention-- rgb[0] is blue, rgb[1] is green, rgb[2] is red.
 *
 * Color space conversion coefficients taken from the excellent
 * http://www.inforamp.net/~poynton/ColorFAQ.html
```

```
 * In his terminology, this is a CCIR 601.1 YCbCr -> RGB.
 * Y values are given for all 4 pixels, but the U (Pb)
 * and V (Pr) are assumed constant over the 2x2 block.
 *
 * To avoid floating point arithmetic, the color conversion
 * coefficients are scaled into 16.16 fixed-point integers.
 * They were determined as follows:
 *
 *       double brightness = 1.0;  (0->black; 1->full scale)
 *       double saturation = 1.0;  (0->greyscale; 1->full color)
 *       double fixScale = brightness * 256 * 256;
 *       int rvScale = (int)(1.402 * saturation * fixScale);
 *       int guScale = (int)(-0.344136 * saturation * fixScale);
 *       int gvScale = (int)(-0.714136 * saturation * fixScale);
 *       int buScale = (int)(1.772 * saturation * fixScale);
 *       int yScale = (int)(fixScale);
 */

/* LIMIT: convert a 16.16 fixed-point value to a byte, with clipping. */
#define LIMIT(x) ((x)>0xffffff?0xff: ((x)<=0xffff?0:((x)>>16)))

/*
 */
static inline void
v4l_copy_420_block (int yTL, int yTR, int yBL, int yBR, int u, int v,
         int rowPixels, unsigned char * rgb, int bits)
{
         const int rvScale = 91881;
         const int guScale = -22553;
         const int gvScale = -46801;
         const int buScale = 116129;
         const int yScale  = 65536;
         int r, g, b;

         g = guScale * u + gvScale * v;
         r = rvScale * v;
         b = buScale * u;

         yTL *= yScale; yTR *= yScale;
         yBL *= yScale; yBR *= yScale;

         if (bits == 24) {
                  /* Write out top two pixels */
                  rgb[0] = LIMIT(b+yTL); rgb[1] = LIMIT(g+yTL); rgb[2] =
LIMIT(r+yTL);
                  rgb[3] = LIMIT(b+yTR); rgb[4] = LIMIT(g+yTR); rgb[5] =
LIMIT(r+yTR);

                  /* Skip down to next line to write out bottom two pixels */
                  rgb += 3 * rowPixels;
                  rgb[0] = LIMIT(b+yBL); rgb[1] = LIMIT(g+yBL); rgb[2] =
LIMIT(r+yBL);
                  rgb[3] = LIMIT(b+yBR); rgb[4] = LIMIT(g+yBR); rgb[5] =
LIMIT(r+yBR);
         } else if (bits == 16) {
                  /* Write out top two pixels */
                  rgb[0] = ((LIMIT(b+yTL) >> 3) & 0x1F) | ((LIMIT(g+yTL) << 3) &
0xE0);
                  rgb[1] = ((LIMIT(g+yTL) >> 5) & 0x07) | (LIMIT(r+yTL) & 0xF8);

                  rgb[2] = ((LIMIT(b+yTR) >> 3) & 0x1F) | ((LIMIT(g+yTR) << 3) &
0xE0);
                  rgb[3] = ((LIMIT(g+yTR) >> 5) & 0x07) | (LIMIT(r+yTR) & 0xF8);
```

```
                    /* Skip down to next line to write out bottom two pixels */
                    rgb += 2 * rowPixels;

                    rgb[0] = ((LIMIT(b+yBL) >> 3) & 0x1F) | ((LIMIT(g+yBL) << 3) &
0xE0);
                    rgb[1] = ((LIMIT(g+yBL) >> 5) & 0x07) | (LIMIT(r+yBL) & 0xF8);

                    rgb[2] = ((LIMIT(b+yBR) >> 3) & 0x1F) | ((LIMIT(g+yBR) << 3) &
0xE0);
                    rgb[3] = ((LIMIT(g+yBR) >> 5) & 0x07) | (LIMIT(r+yBR) & 0xF8);
          }
}

/*
 */
static inline void
v4l_copy_422_block (int yTL, int yTR, int u, int v,
          int rowPixels, unsigned char * rgb, int bits)
{
          const int rvScale = 91881;
          const int guScale = -22553;
          const int gvScale = -46801;
          const int buScale = 116129;
          const int yScale  = 65536;
          int r, g, b;

          g = guScale * u + gvScale * v;
          r = rvScale * v;
          b = buScale * u;

          yTL *= yScale; yTR *= yScale;

          if (bits == 24) {
                    /* Write out top two pixels */
                    rgb[0] = LIMIT(b+yTL); rgb[1] = LIMIT(g+yTL); rgb[2] =
LIMIT(r+yTL);
                    rgb[3] = LIMIT(b+yTR); rgb[4] = LIMIT(g+yTR); rgb[5] =
LIMIT(r+yTR);

          } else if (bits == 16) {
                    /* Write out top two pixels */
                    rgb[0] = ((LIMIT(b+yTL) >> 3) & 0x1F) | ((LIMIT(g+yTL) << 3) &
0xE0);
                    rgb[1] = ((LIMIT(g+yTL) >> 5) & 0x07) | (LIMIT(r+yTL) & 0xF8);

                    rgb[2] = ((LIMIT(b+yTR) >> 3) & 0x1F) | ((LIMIT(g+yTR) << 3) &
0xE0);
                    rgb[3] = ((LIMIT(g+yTR) >> 5) & 0x07) | (LIMIT(r+yTR) & 0xF8);
          }
}

/*
 * convert a YUV420P to a rgb image
 */
int
v4l_yuv420p2rgb (unsigned char *rgb_out, unsigned char *yuv_in,
                    int width, int height, int bits)
{
          const int numpix = width * height;
          const unsigned int bytes = bits >> 3;
          int h, w, y00, y01, y10, y11, u, v;
          unsigned char *pY = yuv_in;
```

```
          unsigned char *pU = pY + numpix;
          unsigned char *pV = pU + numpix / 4;
          unsigned char *pOut = rgb_out;

          if (!rgb_out || !yuv_in)
                    return -1;

          for (h = 0; h <= height - 2; h += 2) {
                    for (w = 0; w <= width - 2; w += 2) {
                              y00 = *(pY);
                              y01 = *(pY + 1);
                              y10 = *(pY + width);
                              y11 = *(pY + width + 1);
                              u = (*pU++) - 128;
                              v = (*pV++) - 128;

                              v4l_copy_420_block (y00, y01, y10, y11, u, v, width, pOut,
bits);

                              pY += 2;
                              pOut += bytes << 1;

                    }
                    pY += width;
                    pOut += width * bytes;
          }
          return 0;
}

/*
 * convert a YUV422P to a rgb image
 */
int
v4l_yuv422p2rgb (unsigned char *rgb_out, unsigned char *yuv_in,
                    int width, int height, int bits)
{
          const int numpix = width * height;
          const unsigned int bytes = bits >> 3;
          int h, w, y00, y01, u, v;
          unsigned char *pY = yuv_in;
          unsigned char *pU = pY + numpix;
          unsigned char *pV = pU + numpix / 2;
          unsigned char *pOut = rgb_out;

          if (!rgb_out || !yuv_in)
                    return -1;

          for (h = 0; h < height; h += 1) {
                    for (w = 0; w <= width - 2; w += 2) {
                              y00 = *(pY);
                              y01 = *(pY + 1);
                              u = (*pU++) - 128;
                              v = (*pV++) - 128;

                              v4l_copy_422_block (y00, y01, u, v, width, pOut, bits);

                              pY += 2;
                              pOut += bytes << 1;

                    }
                    //pY += width;
                    //pOut += width * bytes;
          }
```

```
         return 0;
}

###### Subvehicle/refs/od/od_NetLAN.h #####

/* od_NetLAN.h
 *
 * Object Detection:
 *
 */

#ifndef __OD_NETLAN__H__
#define __OD_NETLAN__H__

#include <netinet/in.h>
#include <sys/socket.h>
#include <pthread.h>

#include "od_common.h"
#include "od_Network.h"


#define OD_NET_LAN_PORT       4385

#define OD_NET_BUFFER_SIZE    4096


typedef struct _od_LANServer {
  pthread_t sthandle;
  pthread_t clients[OD_NET_MAX_CLIENTS];
  int clients_socks[OD_NET_MAX_CLIENTS];
  int clients_state[OD_NET_MAX_CLIENTS];
  int num_clients;
  int listen_socket;
} od_LANServer;



int odCreateLANServer( );
int odDestroyLANServer( );
void *odLANServer_thread( void *args );
void *odLANClient_thread( void *args );

int odLANSend( char *buffer, int size , int client_id );

void _sigiohandler( int arg );

#endif /* __OD_NETLAN__H__ */


/* eof */


###### Subvehicle/refs/od/od_dbase.c #####

/* od_dbase.c
 *
 * ObjectDetection: database section source
 *
 */

#include <stdlib.h>
#include <assert.h>
```

```
#include "malloc_chk.h"
#include "od_dbase.h"



// DO NOT WORRY ABOUT THIS FUNCTION RIGHT NOW
od_VertLink *odAddVert( od_DB *db, int x, int y, int z )
{
  return NULL;
} /* od_VertLink */


// DO NOT WORRY ABOUT THIS FUNCTION RIGHT NOW
int odRemoveVert( od_DB *db, verti_3d *vert )
{
  return OD_OK;
} /* odRemoveVert */


// DO NOT WORRY ABOUT THIS FUNCTION RIGHT NOW
//od_ObjLink *odAddObj( od_DB *db, od_Object *parent /* , specs */ )
//{
//  return NULL;
//} /* odAddObj */
//

int odRemoveObj( od_DB *db, od_Object *obj, od_Object **subobjects )
{
  od_ObjLink *link = (od_ObjLink *)obj;

  // Use assert when checking for non-critical truths. Remember that in each
  // release build these statements will be ignored when compiled.
  assert( db && obj );

  db->OLS.tail->next = link;
  link->next = NULL;
  db->OLS.nclinks++;

  if( obj->prev == NULL ) {
    obj->parentObject->subObject = obj->next;
    if( obj->next ) obj->next->prev = NULL;
    *subobjects = obj->subObject;

    obj->subObject = NULL;
    obj->parentObject = NULL;
    obj->next = NULL;
    obj->probability = 0;
    return OD_OK;
  }

  // not finished

  return OD_OK;
} /* odRemoveObj */

int _odCreateVLinkServer( od_DB *db )
{
  int i;
  od_VertLink *Temp;

  assert(db);
  db->VLS.head = malloc(sizeof(od_VertLink));
```

```
  assert(db->VLS.head != NULL);
  db->VLS.tail = db->VLS.head;
  db->VLS.head->next = NULL;
  db->VLS.nclinks++;
  db->VLS.ntlinks++;
  Temp = db->VLS.head;

  for (i=1; i == OD_INIT_VLS_SIZE; i++)
  {
    Temp->next = malloc(sizeof (od_VertLink));
    assert(Temp->next != NULL);
    Temp = Temp->next;
    Temp->next = NULL;
    db->VLS.nclinks++;
    db->VLS.ntlinks++;
    db->VLS.tail = Temp;
  }//end for
  return 1;

}
/* eof */

int _odDestroyVLinkServer(od_DB *db)
{
  int i;
  od_VertLink *Temp;

  while (db->VLS.head != NULL)
  {
    Temp = db->VLS.head;
    db->VLS.head = db->VLS.head->next;
    free(Temp);
    Temp = NULL;
    db->VLS.nclinks--;
    db->VLS.ntlinks--;
  }

  return 1;
}

###### Subvehicle/refs/od/od_LoadImage.c #####

/* od_LoadImage.c
 *
 * ObjectDetection: load our output image from the camera into an IplImage
 *
 */


#include "od_ImageProcess.h"
#include stdio.h

IplImage *odLoadImage( void *src )
{
  IplImage *dst;

  return dst;
} /* odLoadImage */


/* eof */
```

```
###### Subvehicle/refs/od/od_main.c #####

#include "od_common.h"

int main( )
{
  return 0;
} /* main */

/* eof */

###### Subvehicle/refs/od/od_dbase_test.c #####

#include "od_dbase.h"
#include <stdio.h>

int main( )
{
  int check;
  od_DB *db;

  db->objects = NULL;
  db->verticesDB = NULL;
  //db->VLS = NULL;
  //db->OLS = NULL;

  check = _odCreateVLinkServer(db);
  printf("%d", check);
  check  = _odDestroyVLinkServer(db);
  printf("%d", check);

  return 0;
} /* main */

/* eof */


###### Subvehicle/refs/od/od_ImageProcess.c #####

/* od_ImageProcess.h
 *
 * ObjectDetection: Image processing functions
 *
 */

#define _USE_MATH_DEFINES

#include <assert.h>
#include <stdlib.h>
#include <limits.h>
#include <math.h>

#include "od_ImageProcess.h"


IplImage *odAnalyzeImage( IplImage *source, float thresh, od_DB *maindb )
{
  // Variables -----------------------------------------------------------------
  IplImage       *tmp1, *tmp2, *tmp3, *tmp4, *final;
  CvPoint2D32f   *corners;
  CvMemStorage   *line_storage, *fline_storage;
  CvSeq          *lines, *flines;
  CvPoint        *line;
```

```
int corners_count = 20000;
int i;


// Buffer Initializations ---------------------------------------------------
tmp1 = cvCreateImage( cvSize( source->width, source->height ), IPL_DEPTH_8U, 1 );
tmp2 = cvCreateImage( cvSize( source->width, source->height ), IPL_DEPTH_8U, 1 );
tmp3 = cvCreateImage( cvSize( source->width, source->height ), IPL_DEPTH_32F, 1 );
tmp4 = cvCreateImage( cvSize( source->width, source->height ), IPL_DEPTH_32F, 1 );
final = cvCreateImage( cvSize( source->width, source->height ), IPL_DEPTH_8U, 3 );

assert( tmp1 );
assert( tmp2 );
assert( tmp3 );
assert( tmp4 );
assert( final );

corners = malloc( sizeof( CvPoint2D32f ) * corners_count );

assert( corners );

cvCvtColor( source, tmp1, CV_BGR2GRAY );
cvSmooth( tmp1, tmp2, CV_MEDIAN, 3, 3, 0 );
cvZero( tmp1 );


cvCanny( tmp2, tmp1, (float)thresh, (float)thresh * 1.8, 3 );

// Corner Retrieval ---------------------------------------------------------
cvZero( tmp3 );
cvZero( tmp4 );
cvCopy( tmp1, tmp2, NULL );

cvGoodFeaturesToTrack( tmp1, tmp4, tmp3, corners, &corners_count,
                       .001, 3, NULL, 3, 0, 0.04 );

// Line Retrieval -----------------------------------------------------------
line_storage = cvCreateMemStorage( 0 );

assert( line_storage );

lines = cvHoughLines2( tmp1, line_storage, CV_HOUGH_PROBABILISTIC, 1,
                             CV_PI / 720, 30, 15, 3 );

odAnalyzeLines( lines, maindb );
fline_storage = cvCreateMemStorage( 0 );
odDBtoCvSeq_Overview( maindb, &flines, fline_storage );


// TEST CODE
//lines = flines;

for( i = 0; i < lines->total; i++ ) {
  line = (CvPoint*)cvGetSeqElem( lines, i );
  cvLine( final, line[0], line[1], CV_RGB( 0, 0, 255 ), 1, 8, 0 );
}

// Contour and Polygon Retrival ---------------------------------------------
//CvSeq *contours, *current, *result;

//CvMemStorage *contour_storage = cvCreateMemStorage( 0 );
```

```
  //assert( contour_storage );

  //cvFindContours( tmp2, contour_storage, &contours, sizeof( CvContour ),
  //                CV_RETR_TREE, CV_LINK_RUNS, cvPoint( 0, 0 ));

  //current = contours;
  //while( current ) {
  //  result = cvApproxPoly( current, sizeof( CvContour ), contour_storage,
CV_POLY_APPROX_DP,
  //                         cvContourPerimeter( current ) * 0.010, 0 );

  //  //cout << "Sequence Total: " << result->total << endl;
  //  //CvPoint *data = (CvPoint *)result->first->data;
  //  //for( int j = 0; j < result->total; j++ ) {
  //  //  cout << data[j].x << ", " << data[j].y << endl;
  //  //}

  //  //cvDrawContours( final, result, CV_RGB( 0, 255, 0 ), CV_RGB( 255, 0, 0 ),
  //  //                0, 1, 8, cvPoint( 0, 0 ));
  //  //brShowWindows( source, final );
  //  //cvWaitKey( 0 );

  //  current = current->h_next;
  //}

  // Memory Cleanup ------------------------------------------------------------
  //cvReleaseMemStorage( &contour_storage );
  cvReleaseMemStorage( &line_storage );
  cvReleaseMemStorage( &fline_storage );
  cvReleaseImage( &tmp1 );
  cvReleaseImage( &tmp2 );
  cvReleaseImage( &tmp3 );
  cvReleaseImage( &tmp4 );
  free( corners );

  return final;
} /* odAnalyzeImage */


int odAnalyzeLines( CvSeq *lines, od_DB *db )
{
  int i;
  CvPoint   *line;
  od_Object *current;

  for( i = 0; i < lines->total; i++ ) {
    line = (CvPoint*)cvGetSeqElem( lines, i );

    current = db->objects;
    while( current ) {

      if( current->next == NULL ) break;
      current = current->next;
    }

    // otherwise, insert as a new object

  }

  return OD_OK;
} /* odAnalyzeLines */
```

```
float odVectorAngle2D( CvPoint *lineA, CvPoint *lineB )
{
  float   dotproduct, lengthproducts;
  CvPoint  lineC[2];

  lineC[0].x = lineA[0].x;
  lineC[0].y = lineA[0].y;
  lineC[1].x = lineB[1].x - ( lineB[0].x - lineA[0].x );
  lineC[1].y = lineB[1].y - ( lineB[0].y - lineA[0].y );

  dotproduct = ( (float)lineA[1].x * lineC[1].x ) + ( (float)lineA[1].y * lineC[1].y
);
  lengthproducts = odVectorLength2D( lineA ) * odVectorLength2D( lineC );

  return (float)acos( dotproduct / lengthproducts );
} /* odVectorAngle2D */


int odDBtoCvSeq_Overview( od_DB *db, CvSeq **flines, CvMemStorage *fline_storage )
{
  return OD_OK;
} /* odCreateCVSeq */


float __inline odVectorLength2D( CvPoint *line )
{
  return (float)sqrt( (float)
                      ((( line[1].x - line[0].x ) * (line[1].x - line[0].x )) +
                      (( line[1].y - line[0].y ) * ( line[1].y - line[0].y ))));
} /* odVectorLength2D */


/* eof */

###### Subvehicle/refs/od/od_CVWindowManagement.c #####

/* od_CVWindowManagement.c
 *
 * ObjectDetection: window management for OpenCV
 *
 */

#ifdef HAVE_OPENCV_HIGHGUI_H

#include "od_common.h"
#include "od_ImageProcess.h"


int odCreateWindows( )
{
  cvNamedWindow( OD_WIN_SOURCE, 1 );
  cvNamedWindow( OD_WIN_FINAL, 1 );

  return OD_OK;
} /* odCreateWindows */


void odDestroyWindows( )
{
  cvDestroyWindow( OD_WIN_SOURCE );
  cvDestroyWindow( OD_WIN_FINAL );
} /* odDestroyWindows */
```

```
void odShowWindows( IplImage *source, IplImage *final )
{
  cvShowImage( OD_WIN_SOURCE, source );
  cvShowImage( OD_WIN_FINAL, final );
} /* odShowWindows */

#endif /* HAVE_OPENCV_HIGHGUI_H */

/* eof */


###### Subvehicle/refs/od/od_Camera.h #####

/* od_Camera.h
 *
 * Object Detection: Functions to operate the camera
 *
 */

#ifndef __OD_CAMERA__H__
#define __OD_CAMERA__H__

#include <sys/time.h>

#define OD_PIC_WIDTH                640
#define OD_PIC_HEIGHT               480


// Takes a picture and returns the image buffer uncompressed
char *odCameraTakePicture( struct timeval *time, int shutter_speed, int gain );
// Takes image from camera and writes it to specified file, quality 0 - 100
int odCameraImageToJPEGFile( char *image, char *filename, int quality );
// This will write the raw data of the image to the filename specified
int odCameraImageToFile( char *image, char *filename );
int odCameraMove( int pan, int tilt );


#endif /* __OD_CAMERA__H__ */


/* eof */


###### Subvehicle/refs/od/od_dbase.h #####

/* od_dbase.h
 *
 * ObjectDetection: database section header. Data types and functions related
 * to the maintenance of the database. An emphasis on speed and size is very
 * important since this is designed to work on a low memory system.
 *
 */

#ifndef __OD_DBASE__H__
#define __OD_DBASE__H__


#include "od_common.h"


// Data Types ------------------------------------------------------------------
```

```
typedef struct _od_VertLink {
  verti_3d  vert;
  struct _od_VertLink *next;
} od_VertLink;

typedef struct _od_ObjLink {
  od_Object obj;
  struct _od_ObjLink *next;
} od_ObjLink;


typedef struct _od_VLinkServer {
  od_VertLink   *head, *tail;  // links to head and tail of available links
  int           nclinks;  // The current number of links in the server
  int           ntlinks;  // The total number of links created
} od_VLinkServer;

typedef struct _od_OLinkServer {
  od_ObjLink    *head, *tail;  // links to head and tail of available links
  int           nclinks;  // The current number of links in the server
  int           ntlinks;  // The total number of links created
} od_OLinkServer;


typedef struct _od_DB {
  od_Object       *objects;
  od_VertLink     *verticesDB;

  od_VLinkServer  VLS;
  od_OLinkServer  OLS;
} od_DB;


// The initial size of the vertex and object link servers. Also their increments
// as they increase due to overflow
#define OD_INIT_VLS_SIZE          5000
#define OD_INIT_OLS_SIZE          1000

#define OD_INIT_VLS_INC           500
#define OD_INIT_OLS_INC           100

// Functions ------------------------------------------------------------------

int odInitDB( od_DB *db );
int odDestroyDB( od_DB *db );


/* Creates the link server in the DB. This is a circular linked list. */
int _odCreateVLinkServer( od_DB *db );
/* Used to destroy the links in the link server at the end of the program. */
int _odDestroyVLinkServer( od_DB *db );
/* Used to initialize the links and to resize the link server list depending on
memory needs. If count < 0 then it reduces, > 0 increase. Returns the number
of links added or reduced. Links are added or reduced from the end of the
list */
int _odResizeVLinkServer( od_DB *db, int count );

/* Same as above but for object link server. Almost identical functions */
int _odCreateOLinkServer( od_DB *db );
int _odDestroyOLinkServer( od_DB *db );
int _odResizeOLinkServer( od_DB *db );
```

```
/* Given the xyz coordinates and the type of link to create, a link is obtained
from the link server, set to the appropriate values, and inserted properly in the
vertices database. The pointer to the link just made is returned. NULL on error. */
od_VertLink *odAddVert( od_DB *db, int x, int y, int z );
/* Given a pointer to a vertex, it will remove it from the vertices list and
replace it in the link server. non-zero returned on success. This maintains the
integrity of the list it removes the item from. */
int odRemoveVert( od_DB *db, verti_3d *vert );

/* Same as above but for objects */
od_ObjLink *odAddObj( od_DB *db, od_Object *parent /* , specs */ );
int odRemoveObj( od_DB *db, od_Object *obj, od_Object **subobjects );


#endif /* __OD_DBASE__H__ */


/* eof */

###### Subvehicle/refs/od/od_SearchPath.c #####

/* od_SearchPath.c
 *
 * ObjectDetection: Search path creation and findings
 *
 */

#include "od_SearchPath.h"

int odCreatePath( od_DB *db, od_Path **path )
{
  return 0;
} /* odCreatePath */


/* eof */


###### Subvehicle/refs/od/od_common.h #####

/* od_common.h
 *
 * ObjectDetection: Common Header
 *
 */


#ifndef __OD_COMMON__H__
#define __OD_COMMON__H__


// Constants --------------------------------------------------------------------

#define OD_OK         1
#define OD_ERROR      0


// Data Types -------------------------------------------------------------------

typedef struct _verti_2d {
  int x, y;
  int probability;
  struct _od_Object *owner;
```

```
} verti_2d;

typedef struct _vertf_2d {
  float x, y;
  int probability;
  struct _od_Object *owner;
} vertf_2d;

typedef struct _verti_3d {
  int x, y, z;
  int probability;
  struct _od_Object *owner;
} verti_3d;

typedef struct _vertf_3d {
  float x, y, z;
  int probability;
  struct _od_Object *owner;
} vertf_3d;


typedef struct _od_Line {
  struct _od_Line     *next, *prev;
  struct _od_Object   *owner;

  verti_3d  *pt1, *pt2;

  int     probability;  // if this is not a line and one of the vertices is
                        // theoretical, then this is set to 0
} od_Line;


typedef struct _od_Object {
  struct _od_Object  *subObject;    // link to any subjects owned by this object
  struct _od_Object  *parentObject; // link to parent object

  struct _od_Object  *next, *prev;  // points to next and previous objects

  struct _od_Line    *lines;  // List of Lines contained by the object

  int   probability;
} od_Object;


// Defined by two unit vectors x 1000 with a common center.
// no floating points used due to no fpu
typedef struct _od_Attitude {
  verti_3d    center;
  verti_3d    forward;
  verti_3d    up;
} od_Attitude;


typedef struct _od_GPScoord {
  // whatever is in a gps coordinate
  int remove_me;
} od_GPScoord;


typedef struct _od_HeliState {
  od_Attitude   attitude;
  od_GPScoord   location;
} od_HeliState;
```

```
// Functions ==========================================================================



#endif /* __OD_COMMON__H__ */


/* eof */

###### Subvehicle/refs/od/od_ImageProcess_test.c #####

/* od_ImageProcess_test.c
 *
 * Description: Building Recognition Main
 *
 */

#include <stdio.h>

#include "od_ImageProcess.h"

#ifdef HAVE_OPENCV_HIGHGUI_H

const int   num_files = 7;
const char  *file_list[] = { "img/fig12.jpg", "img/fig11.jpg", "img/fig2.jpg",
                             "img/fig6.jpg", "img/fig4.jpg", "img/fig5.jpg",
                             "img/fig6.jpg" };

int main( )
{
  IplImage *source, *final;
  od_DB maindb;
  int i;

  if( !odCreateWindows( )) {
    printf( "Error creating windows: Aborting\n" );
    return OD_ERROR;
  }

  for( i = 0; i < num_files; i++ ) {
    source = cvLoadImage( file_list[i], 1 );
    if( source == NULL ) {
      printf( "Error Opening Image: %s\n", file_list[i] );
      continue;
    }

    final = odAnalyzeImage( source, 100.0f, &maindb );

    odShowWindows( source, final );
    cvWaitKey( 0 );

    cvReleaseImage( &source );
    cvReleaseImage( &final );
  }

  odDestroyWindows( );

} /* main */

#else
```

```
int main( )
{
  printf( "Not active without highgui\n" );
  return 0;
} /* alternate main */

#endif /* HAVE_OPENCV_HIGHGUI_H */

/* eof */
```

Subvehicle/refs/od/od_SearchPath_test.c

```
/* od_SearchPath_test.c
 *
 * ObjectDetection: Search path test main
 *
 */

#include "od_SearchPath.h"


int main( )
{
} /* main */

/* eof */
```

Subvehicle/refs/od/od_SearchPath.h

```
/* od_SearchPath.h
 *
 * ObjectDetection: Create a search path to find the logo
 *
 */

#ifndef __SEARCH_PATH__H__
#define __SEARCH_PATH__H__

#include "od_common.h"
#include "od_dbase.h"


typedef struct _od_Path {
  od_HeliState     target;

  struct _od_Path  *next;
  struct _od_Path  *prev;
} od_Path;



/* Given a database with the objects contained, this will create a good, idealy
optimal flight path for searching. the head of the path is the first element,
the next target is the next element in the doubly linked list. path may or may
not be allocated, since this function will be called many times. It is also
used to update the path as new information in the db is known.
This returns how many commands are in the path. 0 for any error encountered. */
int odCreatePath( od_DB *db, od_Path **path );
```

```
#endif /* __SEARCH_PATH__H__ */

/* eof */


###### Subvehicle/OMAR/trunk/serial.h #####

#ifndef SERIAL__H__
#define SERIAL__H__

#include <stdlib.h>
#include <sys/types.h>

#define SERIAL_DEVICE   "/dev/ttyS1"
#define SERIAL_BAUD     B115200
#define SERIAL_BUF_SIZE 256
#define SERIAL_CLOSED   0
#define SERIAL_OPEN     1
#define SERIAL_OK       0
#define SERIAL_ERROR    1
#define SERIAL_FATAL    2
#define SERIAL_MAX_PATH 32

class COMAR;

class CSerial {
private:
  COMAR      *parent;
  int       sd,
            state;
  char      *device;
  u_int8_t  *buf,
            buf_head,
            buf_tail;
  pthread_t thread_id;
  pthread_mutex_t state_lock;

  static void *serialThread (void *);
  int       serial_open ( );
  int       serial_close ( );
  int       serial_initialize ( );

public:
  CSerial (COMAR *Parent, const char *Device = NULL);
  ~CSerial ( );
//  void registerCallback(int (*Callback)()  TODO Setup callbacks
  bool operator! ( ) ;


};

#endif // SERIAL__H__


###### Subvehicle/OMAR/trunk/network.h #####


#ifndef NETWORK_H__
#define NETWORK_H__

#include <netinet/in.h>
#include <sys/types.h>
```

```
#include <pthread.h>

#define NET_PORT    60000
#define NET_OK      1
#define NET_ERROR   0
#define NET_OPEN    1
#define NET_CLOSED  0
#define NET_BUF_LEN 4096

class COMAR;

class CNetwork {
private:
  COMAR *parent;
  int sockfd,
      state,
      buf_head,
      buf_tail;
  char *buffer;
  pthread_t thread_id;
  pthread_mutex_t state_lock;
  struct sockaddr_in addr;

  int network_open( );
  int network_close( );
  int network_initialize( );
  static void *networkThread(void *arg);

public:
  CNetwork(COMAR *Parent = NULL);
  ~CNetwork( );
  bool operator! ( );

};

#endif  // NETWORK_H__

###### Subvehicle/OMAR/trunk/client/client.c #####


#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/select.h>
#include <netinet/in.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>
#include <errno.h>

void usage(const char *prgnam) {
  printf(" usage: %s host port\n", prgnam);
}

int main(int argc, char *argv[]) {
  int size,
      port,
      sockfd;
  char  *host, buffer[128];
  //char **addrlist;
  struct sockaddr_in addr;
  struct hostent *server;
```

```
  if (argc != 3) {
    usage(argv[0]);
    return 1;
  }

  host = argv[1];
  port = atoi(argv[2]);

  if (port <= 0 || port > 65535) {
    printf(" error: port out of range (0-65535)\n");
    return 1;
  }

  if ((sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1) {
    printf(" error: failed to allocate socket\n");
    return 1;
  }

  if ((server = gethostbyname(host)) == NULL) {
    printf(" Failed to get host %s (%s)\n", host, strerror(errno));
    return 1;
  }
#if 0
  printf("%s\naka. ", server->h_name);
  addrlist = server->h_aliases;
  size = 0;
  while (addrlist[size] != NULL) printf("%s", addrlist[size++]);
  printf("\naddr type = %d\nh_length = %d\n", server->h_addrtype, server->h_length);
  addrlist = server->h_addr_list;
  size = 0;
  while (addrlist[size] != NULL) printf("%s\n", addrlist[size++]);
#endif
  memset(&addr, 0, sizeof(addr));

  addr.sin_family = AF_INET;
  addr.sin_port = htons(port);
  addr.sin_addr = *((struct in_addr *)*server->h_addr_list);
/*
  if (bind(sockfd, (struct sockaddr *)&addr, sizeof(addr)) == -1) {
    printf("error: failed to bind port %d. (%s)\n", port, strerror(errno));
    return 1;
  }
  */
  if (connect(sockfd, (struct sockaddr *)&addr, sizeof(addr)) != 0) {
    printf(" error: failed to connect to %s (%s)\n", host, strerror(errno));
    return 1;
  }

  printf(" connection open to %s on %d\n", host, port);
  memset(&buffer, 0, sizeof(buffer));

  while (1) {
    printf("> ");
    fflush(stdin);
    fflush(stdout);
    fgets(buffer, sizeof(buffer), stdin);
    fflush(stdin);
    fflush(stdout);
    if (strncmp(buffer, "quit", 4) == 0) break;
    size = strlen(buffer);
    buffer[size-1] = 0;
    printf("read string of size %d [%s]\n", size, buffer);
```

```
    switch(send(sockfd, buffer, size, 0)) {
      case -1:
        perror(" send failed\n");
        return 1;
        break;
      case 0:
        printf(" server disconnected\n");
        return 1;
        break;
      default:
        putc('\n', stdin);
    }
  }

  printf(" bye\n");

  close(sockfd);

  return 0;
}


###### Subvehicle/OMAR/trunk/main.cpp #####

#include <unistd.h>
#include <signal.h>
#include "Logger.h"
#include "omar.h"

int running;

void sigint(int sig) {
  running = 0;
}

int main ( ) {
  COMAR *pO = new COMAR;
  running = 1;
  while (running) {
    signal(SIGINT, sigint);
    sleep(1);
  }
  delete pO;
  return 0;
}

###### Subvehicle/OMAR/trunk/Logger.h #####

#ifndef LOGGER__H__
#define LOGGER__H__

#include <stdio.h>

#define LOG g_Logger.log

typedef enum {DEBUGLOG, INFOLOG, NOTICELOG, WARNINGLOG, ERRORLOG, FATALLOG, NONELOG}
LOGLEVEL;


class CLogger {
  private:
    FILE  *fp;
    int   logLevel;
```

```
    bool  tee;
  public:
    //CLogger ( int level = INFO);
    CLogger (int level = INFOLOG, const char *logfile = NULL, bool Tee = true);
    ~CLogger ( );
    int Close ( );
    inline void setLogLevel ( int level ) { logLevel = level; }
    inline int getLogLevel ( ) { return logLevel; }
    int log(int level, const char *msg, ...);
};

extern CLogger g_Logger;
#endif // LOGGER__H__


###### Subvehicle/OMAR/trunk/serial.cpp #####

#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <fcntl.h>
#include <termios.h>
#include <time.h>
#include <errno.h>
#include <sys/select.h>
#include "Logger.h"
#include "serial.h"

CSerial::
CSerial (COMAR *Parent, const char *Device) {
  // Initialize members
  parent = Parent;
  sd = -1;
  thread_id = 0;
  buf_head = buf_tail = 0;
  pthread_mutex_init(&state_lock, NULL);
  state = SERIAL_CLOSED;

  device = NULL;
  device = new char[SERIAL_MAX_PATH];
  if (device == NULL) {
    LOG(DEBUGLOG, "Failed to allocate device string.");
    return;
  }

  if (Device) strncpy(device, Device, SERIAL_MAX_PATH);
  else strncpy(device, SERIAL_DEVICE, SERIAL_MAX_PATH);

  buf = NULL;
  buf = new u_int8_t[SERIAL_BUF_SIZE];
  if (buf == NULL) {
    LOG(DEBUGLOG, "Failed to allocate serial buffer.");
    return;
  }
  memset(buf, 0, sizeof(buf));
  // Try to startup serial port service
  if (serial_initialize( ) != SERIAL_OK) {
    return;
  }
}

CSerial::
~CSerial ( ) {
```

```
  serial_close();
  if (buf) delete buf;
  if (device) delete device;
}

void * CSerial::
serialThread (void *arg) {

  int selectRetVal = 0;
  u_int32_t timeoutCnt = 0;
  timeval timeout;
  fd_set  rfds, wfds;
  char  buf[101];
  memset(buf, 0, 101);

  // Set thread owner
  CSerial &pSer = *(reinterpret_cast<CSerial *>(arg));

  LOG(DEBUGLOG, "Serial thread started. (tid=0x%08X)", pSer.thread_id);

  // Read until port is closed
  pthread_mutex_lock(&pSer.state_lock);
  while (pSer.state == SERIAL_OPEN) {
    pthread_mutex_unlock(&pSer.state_lock);

    // Setup args for select(). Must be done before each call to select()
    // as it invalidated them.
    timeout.tv_sec = 0;
    timeout.tv_usec = 10000;
    FD_ZERO(&rfds);
    FD_ZERO(&wfds);
    FD_SET(pSer.sd, &rfds);
    FD_SET(pSer.sd, &wfds);

    // Wait for upto 10ms for serial data
    selectRetVal = select(pSer.sd, &rfds, &wfds, NULL, &timeout);

    if (selectRetVal == -1) {
      // Something bad happened
      LOG(FATALLOG,
          "Something terrible has happened to the serial connection! (%d)",
          errno);
      pthread_mutex_unlock(&pSer.state_lock);
      pSer.serial_close();
      pthread_mutex_lock(&pSer.state_lock);
    } else if (selectRetVal == 0) {
      // 10ms has expired, log this if it happens for too long. (1s)
      timeoutCnt++;
      if (!(timeoutCnt % 500))
        LOG(WARNINGLOG, "No activity on serial port in last %us.",
            timeoutCnt/100);
    } else {
      // We have serial data
      timeoutCnt = 0;
      // Read
      read(pSer.sd, buf, 100);
      LOG(NOTICELOG, "Got some serial data.(%s)", buf);
    }

    pthread_mutex_lock(&pSer.state_lock);
  }
  pthread_mutex_unlock(&pSer.state_lock);
```

```
  pthread_exit(NULL);
}

int CSerial::
serial_initialize( ) {
  int ret;

  // Try to open port
  ret = serial_open ( );
  if (ret != SERIAL_OK) return ret; // Failed to open port

  // Try to start service thread
  ret = pthread_create(&thread_id, NULL, serialThread, (void *)this);
  if (ret != 0) {
    // Failed to start thread
    LOG(DEBUGLOG, "Failed to create serial helper thread.");
    serial_close();
    return SERIAL_ERROR;
  }
  return SERIAL_OK;
}

int CSerial::
serial_open ( ) {
  struct termios sdattrs;

  pthread_mutex_lock(&state_lock);

  // Make sure the port isn't already opened
  if (state != SERIAL_CLOSED) {
    pthread_mutex_unlock(&state_lock);
    LOG(ERRORLOG, "Serial port already open!");
    return SERIAL_ERROR;
  }

  // Open serial port
  sd = open(device, 0 | O_RDWR | O_NOCTTY);

  if (sd == -1) {
    // Failed to open serial port
    pthread_mutex_unlock(&state_lock);
    LOG(ERRORLOG, "Failed to open serial port on %s", device);
    return SERIAL_ERROR;
  }

  // Set serial port attributes
  memset(&sdattrs, 0, sizeof(sdattrs));
  sdattrs.c_cflag = 0 | CS8 | CLOCAL | CREAD;
  sdattrs.c_cc[VMIN] = 1;
  cfsetispeed(&sdattrs, SERIAL_BAUD);
  cfsetospeed(&sdattrs, SERIAL_BAUD);
  tcflush(sd, TCIFLUSH);
  tcsetattr(sd, TCSANOW, &sdattrs);

  // Set port as open
  state = SERIAL_OPEN;

  pthread_mutex_unlock(&state_lock);

  LOG(INFOLOG, "Serial opened on %s.", device);

  return SERIAL_OK;
}
```

```
int CSerial::
serial_close ( ) {

  pthread_mutex_lock(&state_lock);
  // Make sure the port isn't already closed
  if (state != SERIAL_CLOSED) {
    // Close port
    close(sd);
    // Stop thread
    state = SERIAL_CLOSED;
    pthread_mutex_unlock(&state_lock);
    // Wait for thread to return
    if (thread_id) {
      pthread_join(thread_id, NULL);
      LOG(DEBUGLOG, "Serial thread stopped. (tid=0x%08X)", thread_id);
    }
    thread_id = 0;
    LOG(INFOLOG, "Serial closed on %s.", device);
  } else {
    pthread_mutex_unlock(&state_lock);
    // Port was already closed
    LOG(DEBUGLOG, "Serial already closed.");
  }

  return SERIAL_OK;
}

bool CSerial::
operator! ( ) {
  bool ret = true;
  pthread_mutex_lock(&state_lock);
  if (state == SERIAL_OPEN) ret = false;
  pthread_mutex_unlock(&state_lock);

  return ret;
}


###### Subvehicle/OMAR/trunk/imageRec.h #####

#ifndef IMAGE_REC__H__
#define IMAGE_REC__H__

class CImageRec {

};

#endif //IMAGE_REC__H__


###### Subvehicle/OMAR/trunk/v4l/pwc-ioctl.h #####

#ifndef PWC_IOCTL_H
#define PWC_IOCTL_H

/* (C) 2001-2003 Nemosoft Unv.    webcam@smcc.demon.nl

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2 of the License, or
   (at your option) any later version.
```

```
   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

/*         This is pwc-ioctl.h belonging to PWC 8.10                        */

/*
   Changes
   2001/08/03  Alvarado   Added ioctl constants to access methods for
                          changing white balance and red/blue gains
   2002/12/15  G. H. Fernandez-Toribio   VIDIOCGREALSIZE
 */

/* These are private ioctl() commands, specific for the Philips webcams.
   They contain functions not found in other webcams, and settings not
   specified in the Video4Linux API.

   The #define names are built up like follows:
   VIDIOC          VIDeo IOCtl prefix
         PWC               Philps WebCam
            G           optional: Get
            S           optional: Set
              ...   the function
 */



/* The frame rate is encoded in the video_window.flags parameter using
   the upper 16 bits, since some flags are defined nowadays. The following
   defines provide a mask and shift to filter out this value.

   In 'Snapshot' mode the camera freezes its automatic exposure and colour
   balance controls.
 */
#define PWC_FPS_SHIFT                16
#define PWC_FPS_MASK                 0x00FF0000
#define PWC_FPS_FRMASK               0x003F0000
#define PWC_FPS_SNAPSHOT    0x00400000



struct pwc_probe
{
         char name[32];
         int type;
};


/* pwc_whitebalance.mode values */
#define PWC_WB_INDOOR                0
#define PWC_WB_OUTDOOR               1
#define PWC_WB_FL           2
#define PWC_WB_MANUAL                3
#define PWC_WB_AUTO                  4

/* Used with VIDIOCPWC[SG]AWB (Auto White Balance).
```

```
   Set mode to one of the PWC_WB_* values above.
   *red and *blue are the respective gains of these colour components inside
   the camera; range 0..65535
   When 'mode' == PWC_WB_MANUAL, 'manual_red' and 'manual_blue' are set or read;
   otherwise undefined.
   'read_red' and 'read_blue' are read-only.
*/

struct pwc_whitebalance
{
         int mode;
         int manual_red, manual_blue;          /* R/W */
         int read_red, read_blue;    /* R/O */
};

/*
   'control_speed' and 'control_delay' are used in automatic whitebalance mode,
   and tell the camera how fast it should react to changes in lighting, and
   with how much delay. Valid values are 0..65535.
*/
struct pwc_wb_speed
{
         int control_speed;
         int control_delay;

};

/* Used with VIDIOCPWC[SG]LED */
struct pwc_leds
{
         int led_on;                           /* Led on-time; range = 0..25000 */
         int led_off;                          /* Led off-time; range = 0..25000  */
};

/* Image size (used with GREALSIZE) */
struct pwc_imagesize
{
         int width;
         int height;
};

/* Defines and structures for Motorized Pan & Tilt */
#define PWC_MPT_PAN                  0x01
#define PWC_MPT_TILT                 0x02
#define PWC_MPT_TIMEOUT              0x04 /* for status */

/* Set angles; when absolute = 1, the angle is absolute and the
   driver calculates the relative offset for you. This can only
   be used with VIDIOCPWCSANGLE; VIDIOCPWCGANGLE always returns
   absolute angles.
 */
struct pwc_mpt_angles
{
         int absolute;               /* write-only */
         int pan;          /* degrees * 100 */
         int tilt;         /* degress * 100 */
         int zoom;         /* N/A, set to -1 */
};

/* Range of angles of the camera, both horizontally and vertically.
   The zoom is not used, maybe in the future...

 */
```

```
struct pwc_mpt_range
{
         int pan_min, pan_max;                /* degrees * 100 */
         int tilt_min, tilt_max;
         int zoom_min, zoom_max;              /* -1, -1 */
};

struct pwc_mpt_status
{
         int status;
         int time_pan;
         int time_tilt;
};


 /* Restore user settings */
#define VIDIOCPWCRUSER               _IO('v', 192)
 /* Save user settings */
#define VIDIOCPWCSUSER               _IO('v', 193)
 /* Restore factory settings */
#define VIDIOCPWCFACTORY    _IO('v', 194)

 /* You can manipulate the compression factor. A compression preference of 0
    means use uncompressed modes when available; 1 is low compression, 2 is
    medium and 3 is high compression preferred. Of course, the higher the
    compression, the lower the bandwidth used but more chance of artefacts
    in the image. The driver automatically chooses a higher compression when
    the preferred mode is not available.
  */
 /* Set preferred compression quality (0 = uncompressed, 3 = highest compression) */
#define VIDIOCPWCSCQUAL              _IOW('v', 195, int)
 /* Get preferred compression quality */
#define VIDIOCPWCGCQUAL              _IOR('v', 195, int)


 /* This is a probe function; since so many devices are supported, it
    becomes difficult to include all the names in programs that want to
    check for the enhanced Philips stuff. So in stead, try this PROBE;
    it returns a structure with the original name, and the corresponding
    Philips type.
    To use, fill the structure with zeroes, call PROBE and if that succeeds,
    compare the name with that returned from VIDIOCGCAP; they should be the
    same. If so, you can be assured it is a Philips (OEM) cam and the type
    is valid.
 */
#define VIDIOCPWCPROBE               _IOR('v', 199, struct pwc_probe)

 /* Set AGC (Automatic Gain Control); int < 0 = auto, 0..65535 = fixed */
#define VIDIOCPWCSAGC                _IOW('v', 200, int)
 /* Get AGC; int < 0 = auto; >= 0 = fixed, range 0..65535 */
#define VIDIOCPWCGAGC                _IOR('v', 200, int)
 /* Set shutter speed; int < 0 = auto; >= 0 = fixed, range 0..65535 */
#define VIDIOCPWCSSHUTTER   _IOW('v', 201, int)

 /* Color compensation (Auto White Balance) */
#define VIDIOCPWCSAWB           _IOW('v', 202, struct pwc_whitebalance)
#define VIDIOCPWCGAWB           _IOR('v', 202, struct pwc_whitebalance)

 /* Auto WB speed */
#define VIDIOCPWCSAWBSPEED  _IOW('v', 203, struct pwc_wb_speed)
#define VIDIOCPWCGAWBSPEED  _IOR('v', 203, struct pwc_wb_speed)

 /* LEDs on/off/blink; int range 0..65535 */
```

```
#define VIDIOCPWCSLED           _IOW('v', 205, struct pwc_leds)
#define VIDIOCPWCGLED           _IOR('v', 205, struct pwc_leds)

  /* Contour (sharpness); int < 0 = auto, 0..65536 = fixed */
#define VIDIOCPWCSCONTOUR   _IOW('v', 206, int)
#define VIDIOCPWCGCONTOUR   _IOR('v', 206, int)

  /* Backlight compensation; 0 = off, otherwise on */
#define VIDIOCPWCSBACKLIGHT _IOW('v', 207, int)
#define VIDIOCPWCGBACKLIGHT _IOR('v', 207, int)

  /* Flickerless mode; = 0 off, otherwise on */
#define VIDIOCPWCSFLICKER   _IOW('v', 208, int)
#define VIDIOCPWCGFLICKER   _IOR('v', 208, int)

  /* Dynamic noise reduction; 0 off, 3 = high noise reduction */
#define VIDIOCPWCSDYNNOISE  _IOW('v', 209, int)
#define VIDIOCPWCGDYNNOISE  _IOR('v', 209, int)

 /* Real image size as used by the camera; tells you whether or not there's a gray
border around the image */
#define VIDIOCPWCGREALSIZE  _IOR('v', 210, struct pwc_imagesize)

 /* Motorized pan & tilt functions */
#define VIDIOCPWCMPTRESET   _IOW('v', 211, int)
#define VIDIOCPWCMPTGRANGE  _IOR('v', 211, struct pwc_mpt_range)
#define VIDIOCPWCMPTSANGLE  _IOW('v', 212, struct pwc_mpt_angles)
#define VIDIOCPWCMPTGANGLE  _IOR('v', 212, struct pwc_mpt_angles)
#define VIDIOCPWCMPTSTATUS  _IOR('v', 213, struct pwc_mpt_status)

#endif

###### Subvehicle/OMAR/trunk/v4l/v4l.h #####

/*
 * v4l.h
 *
 * Copyright (C) 2001 Rasca, Berlin
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#ifndef __V4L_H__
#define __V4L_H__

#define INPUT_DEFAULT       -1
#define NORM_DEFAULT        -1

#ifndef TRUE
#define TRUE 1
#endif
```

```
#ifndef FALSE
#define FALSE 0
#endif

int v4l_set_input (int fd, int input, int norm);
int v4l_check_size (int fd, int *width, int *height);
int v4l_check_palette (int fd, int *palette);
int v4l_mute_sound (int fd);
int v4l_check_mmap (int fd, int *size);
int v4l_yuv420p2rgb (unsigned char *, unsigned char *, int, int, int);
int v4l_yuv422p2rgb (unsigned char *, unsigned char *, int, int, int);
#endif

###### Subvehicle/OMAR/trunk/v4l/v4l.c #####

/*
 * v4l.c
 *
 * Copyright (C) 2001 Rasca, Berlin
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/types.h>
#include <linux/videodev.h>
#include "v4l.h"

#define min(a,b) ((a) < (b) ? (a) : (b))
#define max(a,b) ((a) > (b) ? (a) : (b))

/*
 * set the input and norm for the video4linux device
 */
int
v4l_set_input (int fd, int input, int norm)
{
         struct video_channel vid_chnl;

         if (input != INPUT_DEFAULT || norm != NORM_DEFAULT) {
                  if (vid_chnl.channel != INPUT_DEFAULT)
                           vid_chnl.channel = input;
                  else
                           vid_chnl.channel = 0;
                  vid_chnl.norm = -1;
```

```
                    if (ioctl (fd, VIDIOCGCHAN, &vid_chnl) == -1) {
                              perror ("ioctl (VIDIOCGCHAN)");
                              return -1;
                    } else {
                              if (input != 0)
                                        vid_chnl.channel = input;
                              if (norm != NORM_DEFAULT)
                                        vid_chnl.norm    = norm;
                              if (ioctl (fd, VIDIOCSCHAN, &vid_chnl) == -1) {
                                        perror ("ioctl (VIDIOCSCHAN)");
                                        return -1;
                              }
                    }
          }
          return 0;
}

/*
 * check the size and readjust if necessary
 */
int
v4l_check_size (int fd, int *width, int *height)
{
          struct video_capability vid_caps;

          if (ioctl (fd, VIDIOCGCAP, &vid_caps) == -1) {
                    perror ("ioctl (VIDIOCGCAP)");
                    return -1;
          }
          /* readjust if necessary */
          if (*width > vid_caps.maxwidth || *width < vid_caps.minwidth) {
                    *width = min (*width, vid_caps.maxwidth);
                    *width = max (*width, vid_caps.minwidth);
                    fprintf (stderr, "readjusting width to %d\n", *width);
          }
          if (*height > vid_caps.maxheight || *height < vid_caps.minheight) {
                    *height = min (*height, vid_caps.maxheight);
                    *height = max (*height, vid_caps.minheight);
                    fprintf (stderr, "readjusting height to %d\n", *height);
          }
          return 0;
}

/*
 * check the requested palette and adjust if possible
 * seems not to work :-(
 */
int
v4l_check_palette (int fd, int *palette)
{
          struct video_picture vid_pic;

          if (!palette)
                    return -1;

          if (ioctl (fd, VIDIOCGPICT, &vid_pic) == -1) {
                    perror ("ioctl (VIDIOCGPICT)");
                    return -1;
          }
          vid_pic.palette = *palette;
          if (ioctl (fd, VIDIOCSPICT, &vid_pic) == -1) {
                    /* try YUV420P
                     */
```

```
                    fprintf (stderr, "failed\n");
                    vid_pic.palette = *palette = VIDEO_PALETTE_YUV420P;
                    if (ioctl (fd, VIDIOCSPICT, &vid_pic) == -1) {
                              perror ("ioctl (VIDIOCSPICT) to YUV");
                              /* ok, try grayscale..
                               */
                              vid_pic.palette = *palette = VIDEO_PALETTE_GREY;
                              if (ioctl (fd, VIDIOCSPICT, &vid_pic) == -1) {
                                        perror ("ioctl (VIDIOCSPICT) to GREY");
                                        return -1;
                              }
                    }
          }
          return 0;
}

/*
 * check if driver supports mmap'ed buffer
 */
int
v4l_check_mmap (int fd, int *size)
{
          struct video_mbuf vid_buf;

          if (ioctl (fd, VIDIOCGMBUF, &vid_buf) == -1) {
                    return -1;
          }
          if (size)
                    *size = vid_buf.size;
          return 0;
}

/*
 * mute sound if available
 */
int
v4l_mute_sound (int fd)
{
          struct video_capability vid_caps;
          struct video_audio vid_aud;

          if (ioctl (fd, VIDIOCGCAP, &vid_caps) == -1) {
                    perror ("ioctl (VIDIOCGCAP)");
                    return -1;
          }
          if (vid_caps.audios > 0) {
                    /* mute the sound */
                    if (ioctl (fd, VIDIOCGAUDIO, &vid_aud) == -1) {
                              return -1;
        } else {
           vid_aud.flags = VIDEO_AUDIO_MUTE;
           if (ioctl (fd, VIDIOCSAUDIO, &vid_aud) == -1)
                                        return -1;
        }
    }
          return 0;
}

/*
 * Turn a YUV4:2:0 block into an RGB block
 *
 * Video4Linux seems to use the blue, green, red channel
 * order convention-- rgb[0] is blue, rgb[1] is green, rgb[2] is red.
```

```
 *
 * Color space conversion coefficients taken from the excellent
 * http://www.inforamp.net/~poynton/ColorFAQ.html
 * In his terminology, this is a CCIR 601.1 YCbCr -> RGB.
 * Y values are given for all 4 pixels, but the U (Pb)
 * and V (Pr) are assumed constant over the 2x2 block.
 *
 * To avoid floating point arithmetic, the color conversion
 * coefficients are scaled into 16.16 fixed-point integers.
 * They were determined as follows:
 *
 *       double brightness = 1.0;  (0->black; 1->full scale)
 *       double saturation = 1.0;  (0->greyscale; 1->full color)
 *       double fixScale = brightness * 256 * 256;
 *       int rvScale = (int)(1.402 * saturation * fixScale);
 *       int guScale = (int)(-0.344136 * saturation * fixScale);
 *       int gvScale = (int)(-0.714136 * saturation * fixScale);
 *       int buScale = (int)(1.772 * saturation * fixScale);
 *       int yScale = (int)(fixScale);
 */

/* LIMIT: convert a 16.16 fixed-point value to a byte, with clipping. */
#define LIMIT(x) ((x)>0xffffff?0xff: ((x)<=0xffff?0:((x)>>16)))

/*
 */
static inline void
v4l_copy_420_block (int yTL, int yTR, int yBL, int yBR, int u, int v,
         int rowPixels, unsigned char * rgb, int bits)
{
         const int rvScale = 91881;
         const int guScale = -22553;
         const int gvScale = -46801;
         const int buScale = 116129;
         const int yScale  = 65536;
         int r, g, b;

         g = guScale * u + gvScale * v;
         r = rvScale * v;
         b = buScale * u;

         yTL *= yScale; yTR *= yScale;
         yBL *= yScale; yBR *= yScale;

         if (bits == 24) {
                  /* Write out top two pixels */
                  rgb[0] = LIMIT(b+yTL); rgb[1] = LIMIT(g+yTL); rgb[2] =
LIMIT(r+yTL);
                  rgb[3] = LIMIT(b+yTR); rgb[4] = LIMIT(g+yTR); rgb[5] =
LIMIT(r+yTR);

                  /* Skip down to next line to write out bottom two pixels */
                  rgb += 3 * rowPixels;
                  rgb[0] = LIMIT(b+yBL); rgb[1] = LIMIT(g+yBL); rgb[2] =
LIMIT(r+yBL);
                  rgb[3] = LIMIT(b+yBR); rgb[4] = LIMIT(g+yBR); rgb[5] =
LIMIT(r+yBR);
         } else if (bits == 16) {
                  /* Write out top two pixels */
                  rgb[0] = ((LIMIT(b+yTL) >> 3) & 0x1F) | ((LIMIT(g+yTL) << 3) &
0xE0);
                  rgb[1] = ((LIMIT(g+yTL) >> 5) & 0x07) | (LIMIT(r+yTL) & 0xF8);
```

```
                    rgb[2] = ((LIMIT(b+yTR) >> 3) & 0x1F) | ((LIMIT(g+yTR) << 3) &
0xE0);
                    rgb[3] = ((LIMIT(g+yTR) >> 5) & 0x07) | (LIMIT(r+yTR) & 0xF8);

                    /* Skip down to next line to write out bottom two pixels */
                    rgb += 2 * rowPixels;

                    rgb[0] = ((LIMIT(b+yBL) >> 3) & 0x1F) | ((LIMIT(g+yBL) << 3) &
0xE0);
                    rgb[1] = ((LIMIT(g+yBL) >> 5) & 0x07) | (LIMIT(r+yBL) & 0xF8);

                    rgb[2] = ((LIMIT(b+yBR) >> 3) & 0x1F) | ((LIMIT(g+yBR) << 3) &
0xE0);
                    rgb[3] = ((LIMIT(g+yBR) >> 5) & 0x07) | (LIMIT(r+yBR) & 0xF8);
          }
}

/*
 */
static inline void
v4l_copy_422_block (int yTL, int yTR, int u, int v,
          int rowPixels, unsigned char * rgb, int bits)
{
          const int rvScale = 91881;
          const int guScale = -22553;
          const int gvScale = -46801;
          const int buScale = 116129;
          const int yScale  = 65536;
          int r, g, b;

          g = guScale * u + gvScale * v;
          r = rvScale * v;
          b = buScale * u;

          yTL *= yScale; yTR *= yScale;

          if (bits == 24) {
                    /* Write out top two pixels */
                    rgb[0] = LIMIT(b+yTL); rgb[1] = LIMIT(g+yTL); rgb[2] =
LIMIT(r+yTL);
                    rgb[3] = LIMIT(b+yTR); rgb[4] = LIMIT(g+yTR); rgb[5] =
LIMIT(r+yTR);

          } else if (bits == 16) {
                    /* Write out top two pixels */
                    rgb[0] = ((LIMIT(b+yTL) >> 3) & 0x1F) | ((LIMIT(g+yTL) << 3) &
0xE0);
                    rgb[1] = ((LIMIT(g+yTL) >> 5) & 0x07) | (LIMIT(r+yTL) & 0xF8);

                    rgb[2] = ((LIMIT(b+yTR) >> 3) & 0x1F) | ((LIMIT(g+yTR) << 3) &
0xE0);
                    rgb[3] = ((LIMIT(g+yTR) >> 5) & 0x07) | (LIMIT(r+yTR) & 0xF8);
          }
}

/*
 * convert a YUV420P to a rgb image
 */
int
v4l_yuv420p2rgb (unsigned char *rgb_out, unsigned char *yuv_in,
                    int width, int height, int bits)
{
          const int numpix = width * height;
```

```
	const unsigned int bytes = bits >> 3;
	int h, w, y00, y01, y10, y11, u, v;
	unsigned char *pY = yuv_in;
	unsigned char *pU = pY + numpix;
	unsigned char *pV = pU + numpix / 4;
	unsigned char *pOut = rgb_out;

	if (!rgb_out || !yuv_in)
		return -1;

	for (h = 0; h <= height - 2; h += 2) {
		for (w = 0; w <= width - 2; w += 2) {
			y00 = *(pY);
			y01 = *(pY + 1);
			y10 = *(pY + width);
			y11 = *(pY + width + 1);
			u = (*pU++) - 128;
			v = (*pV++) - 128;

			v4l_copy_420_block (y00, y01, y10, y11, u, v, width, pOut,
bits);

			pY += 2;
			pOut += bytes << 1;

		}
		pY += width;
		pOut += width * bytes;
	}
	return 0;
}

/*
 * convert a YUV422P to a rgb image
 */
int
v4l_yuv422p2rgb (unsigned char *rgb_out, unsigned char *yuv_in,
		int width, int height, int bits)
{
	const int numpix = width * height;
	const unsigned int bytes = bits >> 3;
	int h, w, y00, y01, u, v;
	unsigned char *pY = yuv_in;
	unsigned char *pU = pY + numpix;
	unsigned char *pV = pU + numpix / 2;
	unsigned char *pOut = rgb_out;

	if (!rgb_out || !yuv_in)
		return -1;

	for (h = 0; h < height; h += 1) {
		for (w = 0; w <= width - 2; w += 2) {
			y00 = *(pY);
			y01 = *(pY + 1);
			u = (*pU++) - 128;
			v = (*pV++) - 128;

			v4l_copy_422_block (y00, y01, u, v, width, pOut, bits);

			pY += 2;
			pOut += bytes << 1;

		}
```

```
                    //pY += width;
                    //pOut += width * bytes;
          }
          return 0;
}

###### Subvehicle/OMAR/trunk/Logger.cpp #####

#include <time.h>
#include <stdarg.h>
#include "Logger.h"

static char szLOGLEVEL[][8] = {"DEBUG", "INFO", "NOTICE", "WARNING", "ERROR", "FATAL",
"NONE"};

CLogger g_Logger;

CLogger::
CLogger (int level, const char *logfile, bool Tee) {
  fp = NULL;
  tee = Tee;
#ifdef DEBUG
  logLevel = DEBUGLOG;
#else
  logLevel = level;
#endif

  if (logfile != NULL && level != NONELOG) {
    fp = fopen(logfile, "w");

    if (fp == NULL) {
      fp = stderr;
      log(WARNINGLOG, "Logger failed to open %s. Writing to stderr.", logfile);
    } else {
      log(DEBUGLOG, "Logger started with level %s. Logging to %s.",
szLOGLEVEL[logLevel], logfile);
    }
  } else {
    fp = stderr;
    log(DEBUGLOG, "Logger started with level %s. Logging to stderr.",
szLOGLEVEL[logLevel]);
  }
}

CLogger::
~CLogger ( ) {
  if (fp != NULL) {
    log(DEBUGLOG, "Logger closed.");
    if (fp != stderr) fclose(fp);
  }
  fp = NULL;
}

int CLogger::
log (int level, const char *msg, ...) {
  struct tm *timeStamp;
  time_t t;
  char date[21];
  va_list list;
  if (level < logLevel) return 0;
  va_start(list, msg);
  t = time(NULL);
  timeStamp = localtime(&t);
```

```
  strftime(date, 20, "%Y-%m-%d %H:%M:%S", timeStamp);
  fprintf(fp, "%20s %8s: ", date, szLOGLEVEL[level]);
  vfprintf(fp, msg, list);
  fputc('\n', fp);
  if (tee && fp != stderr) {
    fprintf(stderr, "%20s %8s: ", date, szLOGLEVEL[level]);
    vfprintf(stderr, msg, list);
    fputc('\n', stderr);
  }
  va_end(list);
  return 0;
}



###### Subvehicle/OMAR/trunk/omar.cpp #####

#include "omar.h"
#include "Logger.h"

COMAR::
COMAR( ) {
  pSer = NULL;
  pCam = NULL;
  pNet = NULL;

  if (!*(pSer = new CSerial(this, NULL))) LOG(ERRORLOG,"Serial failed");
  if (!*(pCam = new CCamera(this, NULL))) LOG(ERRORLOG, "Camera failed");
  if (!*(pNet = new CNetwork(this))) LOG(ERRORLOG, "Network failed");
}

COMAR::
~COMAR( ) {
  if (pSer) delete pSer;
  if (pCam) delete pCam;
  if (pNet) delete pNet;
}


###### Subvehicle/OMAR/trunk/messages.h #####

/* messages.h
 *  defines interface with microcontroller
 */

#ifndef MESSAGES__H__
#define MESSAGES__H__



#endif // MESSAGES__H__


###### Subvehicle/OMAR/trunk/camera.cpp #####

/*****************************
 * This code is based off of ../../refs/od/od_Camera.c.
 *****************************
 * Many of the functions are from vidcat.c from the w3cam project
 *
 * Copyright (C) 1998 - 2001 Rasca, Berlin
 *
 * This program is free software; you can redistribute it and/or modify
```

```
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <sys/time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/types.h>
#include <linux/videodev.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <fcntl.h>
#include "camera.h"
#include "Logger.h"

#ifdef __cplusplus
extern "C" {
#include <jpeglib.h>
#include "pwc-ioctl.h"
#include "v4l.h"
}
#endif

CCamera::
CCamera(COMAR *Parent, const char *Device) {
  parent = Parent;
  cd = -1;
  thread_id = 0;
  pthread_mutex_init(&state_lock, NULL);
  state = CAM_CLOSED;
  input = INPUT_DEFAULT;
  norm = NORM_DEFAULT;
  shutter_speed = CAM_SHUT_SPD;
  gain = CAM_GAIN;
  width = CAM_WIDTH;
  height = CAM_HEIGHT;
  depth = CAM_DEPTH;
  device = NULL;
  device = new char[CAM_MAX_PATH];

  if (device == NULL) {
    LOG(DEBUGLOG, "Failed to allocate device string");
    return;
  }

  if (Device) strncpy(device, Device, CAM_MAX_PATH);
  else strncpy(device, CAM_DEVICE, CAM_MAX_PATH);

  if (camera_initialize( ) != CAM_OK) {
    return;
```

```
  }
}

CCamera::
~CCamera( ) {
  camera_close();
  if (device) delete device;
}

int CCamera::
camera_initialize( ) {
  int ret;

  ret = camera_open();
  if (ret != CAM_OK) return ret;

  ret = pthread_create(&thread_id, NULL, cameraThread, (void *)this);
  if (ret != 0) {
    LOG(DEBUGLOG, "Failed to create camera helper thread.");
    camera_close();
    return CAM_ERROR;
  }

  return CAM_OK;
}

int CCamera::
camera_open( ) {

  // Make sure camera isn't open already
  pthread_mutex_lock(&state_lock);
  if (state != CAM_CLOSED) {
    pthread_mutex_unlock(&state_lock);
    LOG(ERRORLOG, "Camera already open!");
    return CAM_ERROR;
  }

  // Open camera
  cd = open(device, O_RDWR);
  if (cd == -1) {
    pthread_mutex_unlock(&state_lock);
    LOG(ERRORLOG, "Failed to open camera on %s.", device);
    return CAM_ERROR;
  }

  // Set camera input to defaults
  if (v4l_set_input(cd, input, norm) == -1) {
    pthread_mutex_unlock(&state_lock);
    LOG(ERRORLOG, "Failed to setup camera input!");
    close(cd);
    cd = -1;
    return CAM_ERROR;
  }

  // Check camera dimesions
  if (v4l_check_size(cd, &width, &height) == -1) {
    pthread_mutex_unlock(&state_lock);
    LOG(ERRORLOG, "Camera failed size check!");
    close(cd);
    cd = -1;
    return CAM_ERROR;
  }
```

```
  state = CAM_OPEN;

  pthread_mutex_unlock(&state_lock);

  LOG(INFOLOG, "Camera opened on %s.", device);

  return CAM_OK;
}

int CCamera::
camera_close( ) {

  pthread_mutex_lock(&state_lock);
  if (state != CAM_OPEN) {
    pthread_mutex_unlock(&state_lock);
    LOG(DEBUGLOG, "Camera already closed!");
    return CAM_ERROR;
  }

  close(cd);
  cd = -1;
  state = CAM_CLOSED;
  pthread_mutex_unlock(&state_lock);

  if (thread_id) {
    pthread_join(thread_id, NULL);
    LOG(DEBUGLOG, "Camera thread stopped. (tid=0x%08X)", thread_id);
    thread_id = 0;
  }

  LOG(INFOLOG, "Camera closed on %s.", device);

  return CAM_OK;
}

void * CCamera::
cameraThread(void *arg) {
#ifdef DEBUG
  int   i = 0;
  char  fn[16];
  FILE *fp = NULL;
#endif
  struct timeval start,stop;
  int size;
  char  *map = NULL,
        *convmap = NULL;
  struct video_mbuf vid_buf;
  struct video_mmap vid_mmap;

  CCamera &C = *(reinterpret_cast<CCamera *>(arg));

  LOG(DEBUGLOG, "Camera thread started. (tid=0x%08X)", C.thread_id);

  vid_mmap.format = VIDEO_PALETTE_YUV420P;
  vid_mmap.frame = 0;
  vid_mmap.width = C.width;
  vid_mmap.height = C.height;

  pthread_mutex_lock(&C.state_lock);

  while (C.state == CAM_OPEN) {
#ifdef DEBUG
    gettimeofday(&start, NULL);
```

```
#endif
    pthread_mutex_unlock(&C.state_lock);

    ioctl (C.cd, VIDIOCGMBUF, &vid_buf);
    if (map == NULL) {
      map = (char *)mmap (0, vid_buf.size, PROT_READ|PROT_WRITE,
            MAP_SHARED, C.cd, 0);
      if ((unsigned char *)-1 == (unsigned char *)map) {
        perror ("mmap()");
        map = NULL;
      }
    }

    if (map != NULL && ioctl (C.cd, VIDIOCMCAPTURE, &vid_mmap) == -1) {
      perror ("VIDIOCMCAPTURE");
      fprintf (stderr, "args: width=%d height=%d palette=%d\n",
               vid_mmap.width, vid_mmap.height, vid_mmap.format);
      munmap (map, vid_buf.size);
      map = NULL;
    }

    if (map != NULL && ioctl (C.cd, VIDIOCSYNC, &vid_mmap.frame) == -1) {
      perror ("VIDIOCSYNC");
      munmap (map, vid_buf.size);
      map = NULL;
    }

    if (map != NULL) {
      size = vid_buf.size;
      if ((convmap = (char *)malloc ( C.width * C.height * C.depth )) != NULL) {
        v4l_yuv420p2rgb ((unsigned char *)convmap,
                         (unsigned char *)map, C.width, C.height, C.depth * 8);
        memcpy (map, convmap, (size_t) C.width * C.height * C.depth);
        free (convmap);
        gettimeofday(&stop, NULL);
        LOG(DEBUGLOG, "Cam loop took %fs.", (float)(stop.tv_sec-
start.tv_sec)+((float)(stop.tv_usec-start.tv_usec)/1000000));
        //LOG(INFOLOG, "Camera snapped image!");
#ifdef DEBUG
        sprintf(fn, "/tmp/test%d.jpg", i++);
        i %= 4;
        if ((fp = fopen(fn, "w")) != NULL) {
          put_image_jpeg(fp, map, C.width, C.height, 100, VIDEO_PALETTE_YUV420P);
          fclose(fp);
        } else {
          LOG(DEBUGLOG, "Failed to write image");
        }
#endif
      }
      convmap = NULL;
    } else {
      LOG(ERRORLOG, "Failed to capture image");
    }

    usleep((1000000 - (CAM_FRAMERATE*CAM_DELAY))/CAM_FRAMERATE);
    pthread_mutex_lock(&C.state_lock);
  }
  pthread_mutex_unlock(&C.state_lock);

  if (map != NULL) munmap(map, vid_buf.size);
  map = NULL;
  pthread_exit(NULL);
}
```

```
bool CCamera::
operator! ( ) {
  bool ret = true;

  pthread_mutex_lock(&state_lock);
  if (state == CAM_OPEN) ret = false;
  pthread_mutex_unlock(&state_lock);

  return ret;
}


#if DEBUG
void
put_image_jpeg (FILE *out, char *image, int width, int height, int quality, int
palette)
{
  int y, x, line_width;
  JSAMPROW row_ptr[1];
  struct jpeg_compress_struct cjpeg;
  struct jpeg_error_mgr jerr;
  char *line;

  line = (char *)malloc (width * 3);
  if (!line)
    return;
  cjpeg.err = jpeg_std_error(&jerr);
  jpeg_create_compress (&cjpeg);
  cjpeg.image_width = width;
  cjpeg.image_height= height;

  cjpeg.input_components = 3;
  cjpeg.in_color_space = JCS_RGB;

  jpeg_set_defaults (&cjpeg);
  jpeg_set_quality (&cjpeg, quality, TRUE);
  cjpeg.dct_method = JDCT_FASTEST;
  jpeg_stdio_dest (&cjpeg, out);


  jpeg_start_compress (&cjpeg, TRUE);
  row_ptr[0] = (JSAMPLE *)line;
  if (palette == VIDEO_PALETTE_GREY) {
    line_width = width;
    for ( y = 0; y < height; y++) {
      row_ptr[0] = (JSAMPLE *)image;
      jpeg_write_scanlines (&cjpeg, row_ptr, 1);
      image += line_width;
    }
  } else {
    line_width = width * 3;
    for ( y = 0; y < height; y++) {
      for (x = 0; x < line_width; x+=3) {
        line[x]   = image[x+2];
        line[x+1] = image[x+1];
        line[x+2] = image[x];
      }
      jpeg_write_scanlines (&cjpeg, row_ptr, 1);
      image += line_width;
    }
  }
  jpeg_finish_compress (&cjpeg);
```

```
  jpeg_destroy_compress (&cjpeg);
  free (line);
}
#endif



###### Subvehicle/OMAR/trunk/slam.cpp #####

#include "slam.h"


###### Subvehicle/OMAR/trunk/camera.h #####

#ifndef CAMERA__H__
#define CAMERA__H__

#include <pthread.h>
#include <stdio.h>

#define CAM_DEVICE    "/dev/video0"
#define CAM_MAX_PATH  32
#define CAM_WIDTH     640
#define CAM_HEIGHT    480
#define CAM_DEPTH     3
#define CAM_SHUT_SPD  52000           // TODO Experiment w/ cam shutter speed
#define CAM_GAIN      33000           // TODO Experiment w/ cam gain
#define CAM_DELAY     82000           // Camera takes 82ms to capture
#define CAM_FRAMERATE 4               // Approximate framerate
#define CAM_OK        1
#define CAM_ERROR     0
#define CAM_OPEN      1
#define CAM_CLOSED    0

class COMAR;


class CCamera {
  COMAR *parent;
  char *device;
  int cd,
      state,
      input,
      norm,
      shutter_speed,
      gain,
      width,
      height,
      depth,
      size;
  pthread_t thread_id;
  pthread_mutex_t state_lock;

  int camera_initialize();
  int camera_open();
  int camera_close();

  static void *cameraThread(void *arg);

public:
  CCamera(COMAR *Parent = NULL, const char *Device = NULL);
  ~CCamera();
  bool operator! ( );
```

```
};

void put_image_jpeg(FILE *out, char *image, int width, int height, int quality, int
palette);

#endif  //CAMERA__H__


###### Subvehicle/OMAR/trunk/slam.h #####

#ifndef SLAM__H__
#define SLAM__H__

class CSLAM {

};

#endif // SLAM__H__


###### Subvehicle/OMAR/trunk/omar.h #####

#ifndef OMAR__H__
#define OMAR__H__

#include "serial.h"
#include "camera.h"
#include "network.h"
#include "imageRec.h"
#include "slam.h"

class COMAR {
  CSerial *pSer;
  CCamera *pCam;
  CNetwork *pNet;


public:
  COMAR( );
  ~COMAR( );

};

#endif //OMAR__H__


###### Subvehicle/OMAR/trunk/imageRec.cpp #####

#include "imageRec.h"


###### Subvehicle/OMAR/trunk/network.cpp #####


#include <stdlib.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/select.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>
#include <errno.h>
#include "Logger.h"
```

```
#include "network.h"

CNetwork::
CNetwork(COMAR *Parent) {
  parent = Parent;
  sockfd = -1;
  state = NET_CLOSED;
  buf_head = buf_tail = 0;
  buffer = NULL;
  pthread_mutex_init(&state_lock, NULL);
  if (network_initialize() != NET_OK) {
    return;
  }

  buffer = new char[NET_BUF_LEN];
  if (buffer == NULL) {
    LOG(DEBUGLOG, "Failed to allocate network receive buffer!");
    network_close( );
    return;
  }
  printf("buffer = %#x\n", (unsigned int)buffer);
}


CNetwork::
~CNetwork( ) {
  printf("buffer = %#x\n", (unsigned int)buffer);
  network_close();
  printf("buffer = %#x\n", (unsigned int)buffer);
  if (buffer != NULL) delete[] buffer;
}

int CNetwork::
network_initialize( ) {
  int ret = NET_OK;

  if ((ret = network_open()) != NET_OK) return ret;

  ret = pthread_create(&thread_id, NULL, networkThread, (void *)this);

  if (ret != 0) {
    LOG(DEBUGLOG, "Failed to start network helper thread");
    network_close();
    return NET_ERROR;
  }

  return NET_OK;
}

int CNetwork::
network_open( ) {

  int optval = 1;

  pthread_mutex_lock(&state_lock);
  if (state != NET_CLOSED) {
    pthread_mutex_unlock(&state_lock);
    LOG(DEBUGLOG, "Network already open");
    return NET_ERROR;
  }

  if ((sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1) {
    pthread_mutex_unlock(&state_lock);
```

```
    LOG(DEBUGLOG, "Network failed to allocate socket. (%s)", strerror(errno));
    return NET_ERROR;
  }

  memset(&addr, 0, sizeof(struct sockaddr_in));
  addr.sin_family = AF_INET;
  addr.sin_port = htons(NET_PORT);
  addr.sin_addr.s_addr = htonl(INADDR_ANY);

  if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval)) == -1) {
    pthread_mutex_unlock(&state_lock);
    LOG(DEBUGLOG, "Network failed to set socket options. (%s)", strerror(errno));
    return NET_ERROR;
  }

  if (bind(sockfd, (struct sockaddr *)&addr, sizeof(addr)) != 0) {
    pthread_mutex_unlock(&state_lock);
    LOG(DEBUGLOG, "Network failed to bind to socket. (%s)", strerror(errno));
    return NET_ERROR;
  }

  state = NET_OPEN;
  pthread_mutex_unlock(&state_lock);

  LOG(INFOLOG, "Network ready on port %d", NET_PORT);

  return NET_OK;
}

int CNetwork::
network_close( ) {

  pthread_mutex_lock(&state_lock);
  if (state == NET_CLOSED) {
    pthread_mutex_unlock(&state_lock);
    LOG(DEBUGLOG, "Network already closed");
    return NET_ERROR;
  }
  close(sockfd);
  sockfd = -1;
  state = NET_CLOSED;
  pthread_mutex_unlock(&state_lock);

  if(thread_id) {
    pthread_join(thread_id, NULL);
    LOG(DEBUGLOG, "Network thread stopped. (tid=0x%08X)", thread_id);
    thread_id = 0;
  }

  LOG(INFOLOG, "Network closed");

  return NET_OK;
}

void *CNetwork::
networkThread(void *arg) {

  int recvSize = 0,
      cSock = -1,
      cLen;
#if 1
  int selectRetVal;
  uint32_t timeoutCnt = 0;
```

```
  fd_set nrfds;
  struct timeval timeout;
#endif
  CNetwork &pN = *(reinterpret_cast<CNetwork *>(arg));

  pthread_mutex_lock(&pN.state_lock);
  if (listen(pN.sockfd, SOMAXCONN) != 0) {
    pthread_mutex_unlock(&pN.state_lock);
    LOG(DEBUGLOG, "Network thread started, but failed listen on socket. (%s)",
        strerror(errno));
    pN.network_close();
    pthread_exit(NULL);
  }

  pthread_mutex_unlock(&pN.state_lock);

  LOG(DEBUGLOG, "Network thread started, listening... (tid=0x%08X)",
      pN.thread_id);

  pthread_mutex_lock(&pN.state_lock);
  while (pN.state == NET_OPEN) {
    pthread_mutex_unlock(&pN.state_lock);
    timeout.tv_sec = 0;
    timeout.tv_usec = 10000;
    FD_ZERO(&nrfds);
    FD_SET(pN.sockfd, &nrfds);
    selectRetVal = select(pN.sockfd+1, &nrfds, NULL, NULL, &timeout);
    switch (selectRetVal) {
      case -1:
        LOG(FATALLOG,
            "Something terrible has happened to the network connection! (%s)",
            strerror(errno));
        pN.network_close();
        break;
      case 0:
        timeoutCnt++;
        if (!(timeoutCnt % 500)) {
          LOG(WARNINGLOG, "No client connected to network in the last %us",
              timeoutCnt/100);
        }
        break;
      default:
        timeoutCnt = 0;
        cSock = -1;
        if ((cSock = accept(pN.sockfd, (struct sockaddr *)&pN.addr, (socklen_t
*)&cLen)) == -1) {
          LOG(ERRORLOG, "accept() failed in network. (%s)", strerror(errno));
        } else {
          LOG(INFOLOG, "Client connected to network");
          pthread_mutex_lock(&pN.state_lock);
          while (cSock >= 0 && pN.state == NET_OPEN) {
            pthread_mutex_unlock(&pN.state_lock);
            timeout.tv_sec = 0;
            timeout.tv_usec = 10000;
            FD_ZERO(&nrfds);
            FD_SET(cSock, &nrfds);
            selectRetVal = select(cSock+1, &nrfds, NULL, NULL, &timeout);
            switch (selectRetVal) {
              case -1:
                LOG(FATALLOG,
                    "Something terrible has happened to the network connection! (%s)",
                    strerror(errno));
                pN.network_close();
```

```
                break;
              case 0:
                timeoutCnt++;
                if (!(timeoutCnt % 500)) {
                  LOG(WARNINGLOG, "No activity from network in the last %us",
                      timeoutCnt/100);
                }
                break;
              default:
                timeoutCnt = 0;
                if ((recvSize = recv(cSock, pN.buffer, NET_BUF_LEN, 0 | MSG_DONTWAIT))
== -1) {
                  LOG(ERRORLOG, "Network recv failed! (%s)", strerror(errno));
                } else if (recvSize == 0) {
                  LOG(INFOLOG, "Network client has disconnected");
                  close(cSock);
                  cSock = -1;
                } else {
                  LOG(DEBUGLOG, "Network received packet. Size %d bytes. [%s]",
recvSize, pN.buffer);
                  // TODO Handle packets
                }
            }
            pthread_mutex_lock(&pN.state_lock);
          }
          pthread_mutex_unlock(&pN.state_lock);
        }
    }
    pthread_mutex_lock(&pN.state_lock);
  }
  pthread_mutex_unlock(&pN.state_lock);

  if (cSock != -1) close(cSock);

  pthread_exit(NULL);
}

bool CNetwork::
operator! ( ) {
  bool ret = true;
  pthread_mutex_lock(&state_lock);
  if (state == NET_OPEN) ret = false;
  pthread_mutex_unlock(&state_lock);

  return ret;
}

###### Subvehicle/Sensor_board/trunk/adc.h #####

#ifndef __MY_ADC__
#define __MY_ADC__

#include <avr/interrupt.h>
#include <avr/io.h>

void adc_init();
uint8_t read_adc();
#endif //__MY_ADC__

###### Subvehicle/Sensor_board/trunk/MControl.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
```

```
#include "MControl.h"
#include <util/delay.h>
#include "i2c.h"
#include "timer.h"


#define gain_prop 1
#define gain_deriv 100

long int heading_error = 0;
long int prev_heading_error = 0;
long int prev_time = 0;
long int curr_time =0;


/**********************
 *
 * Go Foward Function
 * Sets INA for both H-Bridges to 1
 * and sets INB for both to 0
 *
 **********************/

void go_forward ( )
{

  PORTC |= (1<<3) | (1<<5);
  PORTC &= ~(1<<2) & ~(1<<4);

  return;
}


/**********************
 *
 * Go Reverse Function
 * Sets INA for both H-Bridges to 0
 * and sets INB for both to 1
 *
 **********************/

void go_reverse ( )
{
  PORTC |= (1<<2) | (1<<4);
  PORTC &= ~(1<<3) & ~(1<<5);

  return;
}


/**********************
 *
 * Turn Left Function
 * Makes 4,5 go reverse
 * Makes 6,7 go forward
 *
 **********************/

void turn_left ( )
{
  PORTC |= (1<<3) | (1<<4);
  PORTC &= ~(1<<2) & ~(1<<5);
```

```
  return;
}

/**********************
 *
 * Turn Right Function
 * Makes 4,5 go forward
 * Makes 6,7 go reverse
 *
 **********************/

void turn_right ( )
{
  PORTC |= (1<<2) | (1<<5);
  PORTC &= ~(1<<3) & ~(1<<4);

  return;
}

/**********************
 *
 * Brake Function
 *
 **********************/

void brake ( )
{
  PORTC &= ~(1<<2) & ~(1<<3) & ~(1<<5) & ~(1<<4);
  _delay_ms(10);

  return;
}

/********************
 *
 * Update PID function
 *
 * *****************/

int16_t update_PID(uint16_t heading_desired, uint16_t curr_heading)
{
   long int pid;
   long int P = 0;
   long int D = 0;
  long int dt;
  char s[200] ={0};

  curr_time = (timer_cnt * 65535) + TCNT1;

  dt = curr_time - prev_time;

  //calculate heading error for proportional
  heading_error = (long int)heading_desired - (long int)curr_heading;

  P = gain_prop * heading_error;
  D = (gain_deriv * (heading_error-prev_heading_error))/dt;
  pid = P + D;

  _delay_ms(32);
  snprintf(s, 200, "d=%4d h=%4d head_err=%8ld prev_head_err=%8ld dt=%8ld P=%8ld
D=%8ld\r",heading_desired, curr_heading, heading_error, prev_heading_error, dt, P,D);
  put_str(s);
  prev_heading_error = heading_error;
```

```
  prev_time = curr_time;

  return pid;
}


###### Subvehicle/Sensor_board/trunk/MControl.h #####

#ifndef MControl__h__
#define MControl__h__

#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/delay.h>

void go_forward();
void go_reverse();
void turn_left();
void turn_right();
void brake();
int16_t update_PID(uint16_t heading_desired, uint16_t curr_heading);


#endif

###### Subvehicle/Sensor_board/trunk/i2c.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
#include "i2c.h"
#include "uart.h"

#define nop() _delay_us(10)

void i2c_init()
{
  //PORTC = 0x03;  //port0 and port1 internally pulled high
  //TWSR = (0 << TWPS0) & (0 << TWPS1); //prescalar 1 default 0
  TWBR = 8;//(F_CPU / 100000UL - 16) / 2; //0x02; //2 with prescalar 1 should = 100kHz
  hmc_transmit(0x42, 0x57);
  //i2c_transmit(0x3A, 0x20, 0xB7);
}//end init


void send_acquire()
{
  hmc_transmit(0x42, 0x41);
  //ADCSRA |= (1 << ADSC);
}


uint16_t get_compass()
{
 return (hmc_read(0x42));
}

void get_accel(uint8_t addr, uint16_t *data)
{
  data[0] = i2c_read(addr, 0x28);
  nop();
  data[0] += i2c_read(addr, 0x29) << 8;
  nop();
  data[1] = i2c_read(addr, 0x2A);
```

```
  nop();
  data[1] += i2c_read(addr, 0x2B) << 8;
  nop();
  data[2] = i2c_read(addr, 0x2C);
  nop();
  data[2] += i2c_read(addr, 0x2D) << 8;
}

uint8_t get_sonar(uint8_t addr)
{
    //i2c_transmit(addr,0,0x51);
    //_delay_us(70);
    return(i2c_read(addr,3));
}

uint8_t i2c_transmit(uint8_t addr, uint8_t reg, uint8_t data)
{
  if ( send_start() == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if ( send_addr_t(addr) == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if ( send_data(reg) == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if (send_data(data) == ERROR)
  {
    send_stop();
    return ERROR;
  }

  send_stop();
  return OK;
}//end i2c_transmit(addr, data);


uint8_t i2c_read(uint8_t addr, uint8_t reg)
{
  if (send_start() ==ERROR)
  {
    send_stop();
    return ERROR;
  }
  if (send_addr_t(addr) == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if (send_data(reg) == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if (send_repeat() == ERROR)
  {
    send_stop();
```

```
    return ERROR;
  }
  if (send_addr_r(addr) == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if (get_nack_data() == ERROR)
  {
    send_stop();
    return ERROR;
  }

  send_stop();
  return TWDR;
}//end i2c_read(addr, reg);

uint8_t hmc_transmit(uint8_t addr, uint8_t data)
{
  //int status = 0;

  if ( send_start() == ERROR)
  {
    send_stop();
    return ERROR;
  }

  if ( send_addr_t(addr) == ERROR)
  {
    send_stop();
    return ERROR;
  }
  if (send_data(data) == ERROR)
  {
    send_stop();
    return ERROR;
  }

  send_stop();
  return OK;
}//end i2c_transmit(addr, data);

uint16_t hmc_read(uint8_t addr)
{

  uint16_t data = 0;

  if (send_start() == ERROR)
  {
    send_stop();
    return ERROR;
  }

  if (send_addr_r(addr) == ERROR)
  {
    send_stop();
    return ERROR;
  }

  if (get_ack_data() == ERROR)
  {
    send_stop();
    return ERROR;
```

```
  }

  data = TWDR << 8;

  if (get_nack_data() == ERROR)
  {
    send_stop();
    return ERROR;
  }

  data += TWDR;
  send_stop();
  return data;
}//end hmc_read(addr, reg);



/******* send_repeat(); ***********************/
int send_repeat()
{
  TWCR = 0 | (1 << TWSTA) | (1 << TWINT) | (1 << TWEN); //repeat start code

  // wait for repeat start sent
  while ( !(TWCR & (1 << TWINT)));

  // check repeat start sent correct
  if ((TWSR & 0xF8) != TW_REP_START)
    return ERROR; //put_str("REP_START_ERROR\n\r");

  return OK;
}//end send_repeat()



/******* send_start() ******************************/
int send_start()
{
  /* send start */
  TWCR = 0 | (1 << TWINT) | (1 << TWSTA) | (1 << TWEN); //send start condition

  /* wait for start sent */
  while (! (TWCR & (1 << TWINT))); // wait for TWINT flag set.(indicates start sent.

  /* check start sent correct */
  if ((TWSR & 0xF8) != START)
    return ERROR; //put_str("START_ERROR\n\r");

  return OK;
}//end send_start



/****** send_addr_t(uint8_t addr); ********************/
int  send_addr_t(uint8_t addr)
{
  /* send addr */
  TWDR = addr;
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT to start transmission of
addresss.

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT)));
```

```
  /*check status = slave sent/ack received */
  if ((TWSR & 0xF8) != TW_MT_SLA_ACK)
    return ERROR;

  return OK;
}//end send_addr


/****** send_addr_r(uint8_t addr); *********************/
int send_addr_r(uint8_t addr)
{
  /* send addr */
  TWDR = addr | 0x01;

  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWEA); //clear TWINT to start
transmission of addresss. send ACK

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT set. indicates Slave addr sent
and ACK received.

  /*check status = slave sent/ack received */
  if ((TWSR & 0xF8) != TW_MR_SLA_ACK)
    return ERROR; // put_str("ADDR_SENT_ERROR\n\r");

  return OK;
}//end send_addr


/**************************************
 *      int send_data(uint8_t data);
 **************************************/
int send_data(uint8_t data)
{
  /* send data */
  TWDR = data;
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT bit to start data transmission.

  /* wait for ACK */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, ACK
received.

  /* check status = MT_DATA_ACK */
  if ( (TWSR & 0xF8) != TW_MT_DATA_ACK)
    return ERROR; // put_str("DATA_SENT_ERROR\n\r");

  return OK;
}// end send_data



/********** send_stop(); ****************/
void send_stop()
{
  /* send STOP condition */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);

  //return OK;
}



/********* int get_data(); ***************/
```

```
int get_data()
{
  /* get data */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT bit to start data read. send
NACK.

  /* wait for ACK sent */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, NACK sent.

  /* check status = MR_DATA_NACK */
  if ( (TWSR & 0xF8) != TW_MR_DATA_NACK)
    return ERROR; //put_str("DATA_REC_ERROR\n\r");

  return OK;
}
/********* int get_data(); ***************/
int get_nack_data()
{
  /* get data */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN); //clear TWINT bit to start data read. send
NACK.

  /* wait for ACK sent */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, NACK sent.

  /* check status = MR_DATA_NACK */
  if ( (TWSR & 0xF8) != TW_MR_DATA_NACK)
    return ERROR; //put_str("DATA_REC_ERROR\n\r");

  return OK;
}

/********* int get_data(); ***************/
int get_ack_data()
{
  /* get data */
  TWCR = 0 | (1 << TWINT) | (1 << TWEN) | (1 << TWEA); //clear TWINT to start
transmission of addresss. send ACK

  /* wait for ACK sent */
  while ( !(TWCR & (1 << TWINT))); // wait for TWINT flag set = data sent, NACK sent.

  /* check status = MR_DATA_NACK */
  if ( (TWSR & 0xF8) != TW_MR_DATA_ACK)
    return ERROR; //put_str("DATA_REC_ERROR\n\r");

  return OK;
}




###### Subvehicle/Sensor_board/trunk/main.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
#include "i2c.h"
#include "uart.h"
#include "timer.h"
#include "MControl.h"

#define DEBUG 0
```

```
#define STOP 0
#define GO   1
#define LEFT 2
#define RIGHT 3
#define REVERSE 4
#define THRESHOLD 50
#define THROTTLE_THRESH 200
#define SONAR_SIZE 8
#define SC 0xE0
#define SL 0xE6
#define SR 0xE2
#define THRESH_STEP 25


int main()
{
  char s[200];
  uint16_t sonar1[SONAR_SIZE] = {0};
  uint16_t sonar2[SONAR_SIZE] = {0};
  uint16_t sonar3[SONAR_SIZE] = {0};
  unsigned int sonar_sum1 = 0;
  unsigned int sonar_sum2 = 0;
  unsigned int sonar_sum3 = 0;
  uint8_t state = STOP;
  uint8_t i=0;
  uint8_t x=0;
  uint8_t sonar_avg_size = 1;
  int left_speed = 0;
  int right_speed = 0;
  uint8_t throttle = 0xFF;
  int16_t speed_gain = 0;
  int8_t thresh_count=0;
  uint16_t curr_heading=0;
  uint16_t heading_desired=0;
  uint8_t offset=0;


  /* init uart and i2c */
  uart_init();
  timer_init();
  i2c_init();
  sei();

  _delay_ms(32);

  brake();
  timer_start();
  while(1)
  {
    send_acquire();

    /*get data*/
    i2c_transmit(SC,0,0x51);
    _delay_us(80);
    sonar1[i] = i2c_read(SC,2) << 8;
    sonar1[i] |= i2c_read(SC,3);

    i2c_transmit(SL,0,0x51);
    _delay_us(80);
    sonar2[i] = i2c_read(SL,2) << 8;
    sonar2[i] |= i2c_read(SL,3);

    i2c_transmit(SR,0,0x51);
```

```
    _delay_us(80);
    sonar3[i] = i2c_read(SR,2) << 8;
    sonar3[i] |= i2c_read(SR,3);

  /* avg data*/
    i = (i+1) % 8;
    sonar_sum3 = sonar_sum1 = sonar_sum2 = 0;
    for (x=0;x<sonar_avg_size;x++)
    {
       sonar_sum1 += sonar1[x];
       sonar_sum2 += sonar2[x];
       sonar_sum3 += sonar3[x];
    }

    sonar_sum1 /= sonar_avg_size;
    sonar_sum2 /= sonar_avg_size;
    sonar_sum3 /= sonar_avg_size;

    if (sonar_avg_size < 8)
      sonar_avg_size++;

    _delay_ms(7);
    curr_heading = get_compass();
    /*DEBUG print sonar values */
#if DEBUG
    _delay_ms(32);
    snprintf(s, 200, "H=%5d tot=%#4x cnt=%3d d=%4u h=%4u LS=%3x RS=%3x
s=%d\r",speed_gain, throttle, thresh_count, heading_desired, curr_heading, OCR0, OCR2,
state);
    put_str(s);
#endif

/********* state machine start**************/

    /* if state = GO, set speeds and direction = forward*/
    if (state == GO)
    {

      /* throttle */
      if (sonar_sum1 > (THROTTLE_THRESH - offset) && sonar_sum2 > (THROTTLE_THRESH-
offset) && sonar_sum3 > (THROTTLE_THRESH-offset))
      {
        if (thresh_count < 5)
        {
          throttle *= 7;
          throttle >>= 3;
          offset += THRESH_STEP;
          thresh_count++;
        }
      }
      else
      {
        if (thresh_count > 0)
        {
          throttle *= 8;
          throttle /= 7;
          throttle ++;
          offset -= THRESH_STEP;
          thresh_count--;
        }
      }

      speed_gain = update_PID(heading_desired, curr_heading);
```

```
  //if neg gain, -left +right, if pos gain, +left -right
  left_speed = throttle + speed_gain/2;
  right_speed = throttle - speed_gain/2;

  if (left_speed < 0)
    left_speed = 0;
  else if (left_speed > 0xFF)
    left_speed = 0xFF;
  if (right_speed < 0)
    right_speed = 0;
  else if (right_speed > 0xFF)
    right_speed = 0xFF;

  //OCR0=left side, OCR2=right side
  OCR0 = (uint8_t)left_speed;
  OCR2 = right_speed;

  go_forward();

  /*check for objects. if any sonar detects object go to state = REVERSE*/
  if (sonar_sum1 < THRESHOLD || sonar_sum2 < THRESHOLD || sonar_sum3 < THRESHOLD)
  {
    brake();
    _delay_ms(10);
    state = REVERSE;
  }
}
/*go_reverse for 10 ms. if S2 detects and S3 does not turn_right, else turn_left*/
else if (state == REVERSE)
{
  OCR0 = 0xFF;
  OCR2 = 0xFF;
  go_reverse();
  _delay_ms(10);

  if (sonar_sum2 < sonar_sum3)
    state = RIGHT;
  else
    state = LEFT;
}
/*state = RIGHT, or LEFT. Turn left or right until no object. Then go*/
else if (state == RIGHT)
{
   if (sonar_sum1 < THRESHOLD+10 || sonar_sum2 < THRESHOLD+10)
   {
      OCR0 = 0xFF;
      OCR2 = 0xFF;
      turn_right();
   }
   else
   {
     brake();
     state = STOP;
   }
}
else if (state == LEFT)
{
  if (sonar_sum1 < THRESHOLD+10 || sonar_sum3 < THRESHOLD+10)
  {
      OCR0 = 0xFF;
      OCR2 = 0xFF;
      turn_left();
```

```
       }
       else
       {
         brake();
         state = STOP;
       }
    }
    else if (state == STOP)
    {
      if (sonar_sum1 < THRESHOLD || sonar_sum2 < THRESHOLD || sonar_sum3 < THRESHOLD)
      {
        brake();
        _delay_ms(10);
        state = REVERSE;
      }
      else
      {
        hmc_transmit(0x42, 0x41);
        brake();
        _delay_ms(32);
        heading_desired = get_compass();
        state = GO;
      }

    }
  }
 return(0);
}//end main


###### Subvehicle/Sensor_board/trunk/uart.c #####

#include <avr/interrupt.h>
#include <avr/io.h>
#define BAUD 12
#include "uart.h"

#define NULL (void*)0

/* UART variables */
static volatile uint8_t tx_read;
static volatile uint8_t tx_write;
static volatile uint8_t tx_buff[TX_BUFF_SIZE];
static volatile uint8_t rx_head;
static volatile uint8_t rx_tail;
static volatile uint8_t rx_buff[TX_BUFF_SIZE];

void uart_init()
{
  /* Enable the UART for sending and receiving */
  sbi(UCSRB, RXEN);
  sbi(UCSRB, TXEN);

  tx_read = tx_write = 0;

  /* Set baud rate registers */
  UBRRH = (((uint16_t)BAUD) & 0xff00) >> 8;
  UBRRL = ((uint16_t)BAUD) & 0xff;

  /* Enable Rx interrupt */
  sbi(UCSRB, RXCIE);
}
```

```
/*UART rc interrupt */
SIGNAL(SIG_UART_RECV)
{
  uint8_t tmp_head;

  tmp_head = rx_head + 1 % RX_BUFF_SIZE;

  /*if rx full*/
  while(tmp_head == rx_tail);

  rx_buff[rx_head] = UDR;

  rx_head = tmp_head;
}

uint8_t get_char()
{
  uint8_t tmp_tail;

  /*rx buffer empty*/
  if(rx_head == rx_tail)
    return((char)NULL);

  tmp_tail = rx_tail;
  rx_tail = (rx_tail + 1) % RX_BUFF_SIZE;

  return rx_buff[tmp_tail];
}

uint8_t get_str(char *buff, uint8_t size)
{
  uint32_t i = 0;

  while ( (buff[i] = get_char()) != (char)NULL && i < size)
    i++;

  return i;
}


/* UART tx interrupt */
SIGNAL(SIG_UART_DATA)
{
  /* Tx buffer is empty */
  if(tx_read == tx_write)
  {
    cbi(UCSRB, UDRIE);
    return;
  }

  UDR = tx_buff[tx_read];

  tx_read = tx_read + 1 >= TX_BUFF_SIZE ? 0 : tx_read + 1;
}

void put_char(uint8_t c)
{
  uint8_t tmp_write;

  tmp_write = tx_write + 1 >= TX_BUFF_SIZE ? 0 : tx_write + 1;

  /* Buffer is full */
```

```
  if(tmp_write == tx_read)
  {
    sbi(UCSRB, UDRIE);
    return;
  }


  tx_buff[tx_write] = c;
  tx_write = tmp_write;

  /* Enable Tx interrupt to start sending */
  sbi(UCSRB, UDRIE);

  return;
}

void put_short(uint16_t s)
{
  put_char(((uint8_t)s & 0xFF));
  put_char((uint8_t)(s >> 8));
}

void put_str(char *str)
{
  uint32_t i = 0;

  while (str[i] != (char)NULL)
  {
    put_char(str[i]);
    i++;
  }
}


###### Subvehicle/Sensor_board/trunk/adc.c #####

#ifndef ADC__H__
#define ADC__H__

#include <avr/interrupt.h>
#include <avr/io.h>
#include "adc.h"

void adc_init(void)
{
  ADCSRA = 0x00;
  PORTA = 0x00; DDRA = 0x00; //port a is input
  ADMUX = 0xE0; //port0 enabled
  ADCSRA = 0x83; //enabled.start.auto trigger.interrupt flag. interupt
enabled.prescalars
 // SFIOR = 0x00; //free running mode, set auto trigger to interupt flag
 // ADCSRA |= (1 << ADEN);
 // ADCSRA |= (1 << ADSC);
 // return;
}

uint8_t read_adc(void)
{
  return ADCH;
}

#endif // ADC__H__
```

```
###### Subvehicle/Sensor_board/trunk/uart.h #####

#ifndef __MY_UART__
#define __MY_UART__

#include <avr/interrupt.h>
#include <avr/io.h>

#define TX_BUFF_SIZE 256
#define RX_BUFF_SIZE 256


#ifndef BAUD
/* If different baud rate is desired do #define BAUD ??
   BEFORE #include "uart.h" in your source */
#define BAUD 51 /* 38400 baud @ 8MHz normal speed */
#endif

#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif

/* Initialize UART, MUST BE CALLED */
void uart_init();
/* Write 8-bit value to UART */
void put_char(uint8_t c);
/* Write 16-bit value to UART */
void put_short(uint16_t s);
/* Write string to UART */
void put_str(char *str);
/* get 8-bit value from UART */
uint8_t get_char();
/* get string from UART */
uint8_t get_str(char*, uint8_t);

#endif /* __MY_UART__ */

###### Subvehicle/Sensor_board/trunk/timer.h #####

#ifndef _timer_h_
#define _timer_h_

#define CLOCK ((unsigned) CLOCK_SPEED)

extern uint16_t timer_cnt;

void timer_start();
void timer_stop();
void timer_reset();


void timer_init(void);


#endif //_timer_h_

###### Subvehicle/Sensor_board/trunk/timer.c #####

#include <avr/interrupt.h>
```

```
#include "timer.h"

uint16_t timer_cnt;

void timer_init(void)
{
  DDRC = 0xFF;     //enable PC2-7 as logic level outputs for motorcontroller
  DDRB |= (1<<3);  //enable PB3 as output for PWM signal
  DDRD |= (1<<7);  //enable PD7 as output for PWM signal
         //Timer0 @ clk0: Motor Controller
         TCCR0 = 0x6A;  //Set Timer/Counter0 to fast PWM mode
                        //Set output compare to asserted state
                        //Prescaler factor to set PWM freq at 3.9KHz
                         //max PWM freq is 20KHz
         OCR0 = 0xFF;   //set output compare register to ?? Duty cycle

         TCCR2 = 0x6A;  //Set Timer/Counter2 to fast PWM mode
                         //Set output compare to asserted state
                         //Prescaler factor to set PWM at 3.9KHz
         OCR2 = 0xFF;   //Set OC reg to something

  TIMSK = 0x04;  //enable timer overflow interrupt

  timer_cnt = 0;

}

void timer_start()
{

  TCCR1A = 0x0C;  //normal mode
  TCCR1B = 0x05;  //set freq to

}

void timer_stop()
{

  TCCR1A = 0X00;
  TCCR1B = 0X00;

}

void timer_reset()
{
  TCNT1H = 0x00;
  TCNT1L = 0x00;
  timer_cnt = 0;

}

ISR(TIMER1_OVF_vect)
{

  timer_cnt++;

}


###### Subvehicle/Sensor_board/trunk/i2c.h #####

#ifndef I2C__H__
#define I2C__H__
```

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>
#include <util/twi.h>

#ifndef F_CPU
#define F_CPU 800000UL
#endif

#include <util/delay.h>

#define START_ERROR 1
#define ADDR_SENT_ERROR 2
#define DATA_SENT_ERROR 3
#define ADDR_REC_ERROR 4
#define DATA_REC_ERROR 5
#define DATA_NACK_REC_ERROR 6
#define SENT_OK 7
#define OK    0
#define ERROR 0xFF
#define START 0x08

void i2c_init();

uint16_t get_compass();
void get_accel(uint8_t addr, uint16_t *data);
inline void send_acquire();
uint8_t get_sonar(uint8_t addr);

uint8_t hmc_transmit(uint8_t addr, uint8_t data);
uint16_t hmc_read(uint8_t addr);
uint8_t i2c_transmit(uint8_t addr, uint8_t reg, uint8_t data);
uint8_t i2c_read(uint8_t addr, uint8_t reg);

/******/
int send_repeat();
int send_start();
void send_stop();
int send_addr_t(uint8_t addr);
int send_addr_r(uint8_t addr);
int send_data(uint8_t data);
int get_data();
int get_ack_data();
int get_nack_data();
/*****/
#endif //I2C__H__
```

Appendix G: FMECA Worksheet

Table G-1 - FMECA Power Subsystem

#	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality
A1	**Output = 0V**	**-Battery fails -LDO fails**	**-Doesn't run**	**-Power LED off**	**Medium**
A2	**Output > 5V**	**-LDO fails**	**-Potential damage to components**	**-Smoke -Excess heat**	**High**
A3	**Input short**	**-LDO fails -Contact of conductive surface with PCB**	**-Component Damage -User injury -Fire**	**-Battery explodes**	**High**
A4	**Discrete component failure**	**-short -regulator failure**	**-Unstable voltage outputs**	**-LED off/fluctuating -Sporadic behavior**	**Low/Medium**

Table G-2 - FMECA Drive Subsystem

#	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality
B1	Windings fail	-Insulation fails -Windings break	-Motor(s) stop	-Vehicle stops -Possible smoke	Medium
B2	Brushes fail	-Wear -High input current	-Motor(s) stop	-Vehicle stops -Possible smoke	Medium

Table G-3 - FMECA Software Subsystem

#	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality
C1	No communication with Gumstix	-Failed interconnect -VCC out of range -Dead port pins	-Failure to map room -Failure to determine new heading after obstacle detected	-Vehicle stops after detecting obstacle	Medium
C2	No communication with sensors	-Failed interconnect -VCC out of range -Dead port pins	-Failure to map room -Failure to detect obstacles	-Vehicle crashes -Fire	High
C3	Failed pushbutton	-object holding pushbutton down	-Microcontroller to be placed in reset mode	-OMAR would stop running	Medium

Table G-4 - Sensor Subsystem

#	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality
D1	Magnetometer Failure	-VCC out of range – Magnetic interference	-Failure to determine compass heading -Failure to drive straight -Failure to determine new direction	-Not driving straight -Failure to turn in correction direction	Low
D2	Sonar Failure	-VCC out of range -False 40 kHz sound interference	-failure to detect objects and walls	-Crash into walls/objects -Possible battery explosion -Possible fire	High
D3	IR Failure	-VCC out of range -Incorrect ADC AVCC -Ambient light interference	-Failure to map room	-No room mapping -Failure to navigate room properly	Low
D4	Accelerometer Failure	-VCC out of range	-Failure to determine position in room	-False room map	Low
D5	Pull resistor failure	-VCC out of range, short	-All sensors fail	-Omar would stop	Medium
D6	Level translators fail	-VCC out of range -Short	-3.3V sensor would fail	-Omar would stop	Medium
D7	ADC inductor fails	-VCC out of range -Short -Bad ADC input value	-IR sensors would fail	-Room mapping would fail	Low